UNIVERSIDADE DE LISBOA
Faculdade de Ciências
Departamento de Informática

Citation Impact Discerning Self-Citation: Improving User Interaction

Tiago Alberto Paiva de Sousa

MESTRADO EM ENGENHARIA INFORMÁTICA
Especialização em Sistemas de Informação

2011

UNIVERSIDADE DE LISBOA
Faculdade de Ciências
Departamento de Informática

Citation Impact Discerning Self-Citation: Improving User Interaction

Tiago Alberto Paiva de Sousa

PROJECTO

Projecto orientado pelo Prof. Doutor Francisco José Moreira Couto
e co-orientado pelo Prof. Doutor Paulo Jorge Esteves Veríssimo

MESTRADO EM ENGENHARIA INFORMÁTICA
Especialização em Sistemas de Informação

2011

Acknowledgments

I would like to thank my advisors for the help, trust and guidance provided, *LaSIGE* for providing me a great environment to work and a scholarship, co-workers for their companionship and sharing their knowledge with me and everyone who, direct or indirectly, helped me doing this thesis.

To my family, who always offered me love and support.

Resumo

CIDS (Citation Impact Discerning Self-Citations) é uma ferramenta web que usa o Google Scholar como repositório dados de publicações, para calcular o impacto de citações das mesmas. Existem outros repositórios de dados que poderiam ter sido usados, como por exemplo, Scopus e Web of Science, mas uma vez que têm uma cobertura de publicações inferior à do Google Scholar, o CIDS usa este último.

Muitas métricas foram propostas para medir o impacto das citações, tais como o h-index e o g-index. Essas métricas têm em conta um conjunto de publicações que normalmente pertencem a um autor e, no caso do h-index e g-index, produzem um número discreto. Este número é uma medida quantitativa do impacto das citações feitas por um autor. Para automatizar o cálculo das métricas, várias ferramentas foram desenvolvidas, tais como a HPP, o Microsoft Academic e o CIDS. O CIDS diferencia-se dos outros por discernir auto-citações. De acordo com o HPP, uma auto-citação é uma citação em que pelo menos um dos autores de uma publicação é também autor da publicação citada, que é precisamente a definição utilizada pelo CIDS. A versão do CIDS (3.0) que me foi entregue, embora fosse mais rápida que a versão antecessora e com a possibilidade de filtrar publicações, não permite que um utilizador tenha em simultâneo mais de um resultado de impacto de citações, isto é, se o utilizador tem dois grupos de publicações e gostava de ver o resultado do impacto de citações calculado para cada um desses conjuntos, tal não é possível. Essa versão também não permite a filtragem de citações de publicações, não tem um mecanismo de cálculo do impacto de citações de grupos usável, e também não funciona no navegador Internet Explorer, um dos browsers mais populares na internet.

O meu objetivo com este trabalho foi criar uma nova versão do CIDS (3.1) que superasse alguns dos problemas acima descritos e ainda ter mais algumas funcionalidades. As funcionalidades projectadas são as seguintes: a atribuição de uma área privada para o utilizador para que possa consultar o progresso das suas interrogaões de pesquisa do Google Scholar, o utilizador com mais controle sobre seus resultados finais, ajudar o utilizador em tarefas repetitivas através da detecção de padrões do seu uso, e corrigir o CIDS para que possa ser executado em todos os navegadores de internet (também apelidado de compatibilidade *cross-browser*). Uma funcionalidade já existente no CIDS anterior (3.0), a análise bibliométrica de um grupo de autores, também seria renovada e extendida, ficando mais automatizada e permitindo ao utilizador poder ser notificado quando os seus

resultados estivessem prontos para visualização.

Após a realização do trabalho, algumas das funcionalidades propostas não chegaram a ser desenvolvidas: compatibilidade cross-browser (embora tenha havida esforço nesse sentido) e a ajuda em tarefas repetitivas, devido a restrições no tempo de desenvolvimento. Ainda assim, a implementação das restantes funcionalidade propostas tornou, na minha opinião, o CIDS numa ferramenta mais capaz e fácil de usar. Através de um esforço extra, o CIDS é agora uma ferramenta que consegue saber quando é que o Google Scholar o bloqueia e como reagir a esse evento (ligando/desligando o acesso, conforme a resposta obtida). Também permite filtrar citações, bem como alterar o seu tipo (de citação própria para não-própria e vice-versa), o que permite ao autor corrigir dados erróneos que possam advir do repositório de publicações do Google Scholar. E, para além de permitir ao utilizador poder ter várias queries de pesquisa em simultâneo, devido a uma reestruturação efectuada, a análise de grupo de autores, que outrora necessita de três passos incómodos para ser realizada, agora necessita apenas de um, e ainda notifica o utilizador quando é que essa análise de grupo termina.

Palavras-chave: Citação, Bibliometria, Impacto, CIDS, h-index

Abstract

CIDS (Citation Impact Discerning Self-Citation) is a web tool that uses Google Scholar as a publication database, to calculate citation impact. Others databases also exist, such as Scopus and Web of Science, but their coverage is lower than the coverage of Google Scholar, so CIDS uses the latter. Many metrics were proposed to measure citation impact, such as h-index and g-index.

These metrics take into account a set of publications, usually belonging to one author, and, in case of h-index and g-index, output a single discrete number. This number is a quantitative measure of citation impact. To automate multiple metrics calculus, several tools were developed, such as HPP, Microsoft Academic and CIDS. CIDS differentiates itself from the others, by discerning self-citations. According to HPP, self-citation occurs when at least one of the publication's authors is also author of the cited publication. The version of CIDS that was delivered to me, albeit faster than its predecessor and with a manual publication filtering process available, did not allow a user to have more than one citation impact result, that is, if the user had two groups of publications and wanted citation to be impact calculated for each one, that was not possible. It also does not allow filtering citations of publications, does not allow several queries calculus simultaneously for one user nor it works on Internet Explorer, a popular web browser. My work objective was to create a new version of CIDS that overcome the issues previously mentioned and with more functionalities. These new functionalities consisted in assigning a private area to the user for his queries, user having more control over his final results, help the user in repetitive tasks by detecting patterns in his usage and fix CIDS so that it can execute on all major browsers (cross-browser compatibility). Previous CIDS's team functionality would also be enhanced to help the user performing group analysis faster. Even though some of the functionalities stated before - cross-browser compatibility and the help on repetitive tasks - were not implemented, the remaining ones were and, in my opinion, contributed to a better tool, perhaps a reference in citation impact tools.

Keywords: Citation, bibliometrics, Impact, CIDS, h-index

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

The increasing use of citation impact for evaluation and comparison, not only of individual researchers but also of institutions, universities and even countries motivated the development of bibliometrics, which are becoming an easy and balanced way to compare and rank scientists and research units.

To facilitate the visualization and calculation of these bibliometrics, several tools were developed, such as HPP [1], Microsoft Academic [2] and CIDS[3] (Citation Impact Discerning Self-citations). However, only one of them, *CIDS*, previously developed in my research lab (LASIGE), allows visualization of those bibliometrics while discerning self-citations. Since the number of self-citations can reach up to 30% of the total number of citation count, it is important to, at least, be able to discern them when calculating the bibliometrics.

In the past, a couple of versions of *CIDS* were developed, each one addressing a specific issue for improving the data retrieval from interfacing with *Google Scholar*, its publication and citation database. One of those versions, 3.0, created a mechanism that increased the rate of requests to *Google Scholar*, which greatly diminished the user waiting period in *CIDS*. In spite of its distinct feature - discerning self-citations - version 3.0 still has some limitations in user interactivity. For instance, even though it is possible to filter publications, the same is not true with citations, which, after all, are of greater importance for this subject. Also, users cannot perform two search queries simultaneously; currently, the user has to wait for the first one to finish, in order to begin the second one. It also has some security problems, since it does little input sanitation. Moreover, it does not work on a major web-browser, Internet Explorer. A critical feature was also left incomplete in CIDS, the group analysis. This functionality enables a user to calculate and visualize bibliometrics results for a group of people (i.e. research units), instead of just one person.

[1] http://www.harzing.com/

[2] http://academic.research.microsoft.com/

[3] http://cids.di.fc.ul.pt/

These limitations are somehow very restrictive and I believe if they were fixed, it would make CIDS a richer tool and perhaps a world-wide reference when discerning self-citations is required.

1.2 Objectives

To address the problems described above, a new CIDS version - 3.1 - is to be created, which is the main objective of this work, and it constitutes the developing of the following functionalities:

User profile: A private area where a user can perform several search queries running at the same time, each one with its own bibliometrics results (currently, only one query is possible). It will also be possible to visualize the progress of those search queries;

Data validation: Allows the user to validate and correct bibliometrics results, by enabling manual filtering citations of publications;

Data verification: Automatic filtering publications and citations by analyzing citation patterns using data mining and cleaning techniques;

Group analysis: A user will be able to manage one or more groups of people, each one (person) with a single query. A user will also be able to add or remove users from groups. This is an extension on previous CIDS's team/upload feature. This step will also be more automated, freeing user from having to wait for each step involved in these operation.

Security: Since CIDS receives user input, security measures are required to prevent a malicious user to corrupt data or gain access to other users' information. Whereas previous version of CIDS does not perform any input filtering, the new features in new CIDS version will;

Cross-browser compatibility: Internet Explorer, one of the major web-browsers known to general population, is currently not supported by CIDS due to Javascript incompatibility, but the new features will be supported.

1.3 Methodology

Before performing the functionalities mentioned in the objectives, I have first analyzed how previous CIDS operated, by sketching work-flows of how different technologies present in CIDS interacted with each other. Only after thinking that I had grasped its inner-workings, I started thinking about how to fulfil the objectives, described next:.

User Profile This objective was fulfilled by first performing a rationale on how it would be implemented on existing system structure and then creating a user access mechanism composed of three modules: *registration*, *login*, and *reset password*.

The rationale done was required because current CIDS structure did not allow a user to have multiple queries simultaneously but for this functionality, it was required. The approach taken was keeping existing CIDS structure and extend it, rather than trying to modify it, because the latter choice was proved to not work.

The *registration* module was done for users to become known to the system, so that they could *login* and access the restricted area afterwards. The *reset password* was done to enable users to be able to login again after forgetting their passwords. Each of these modules required new interfaces to interact with user. After creating this mechanism, I then integrated it with existing code, to enable user navigating between old and new interfaces seamlessly;

Data validation This objective was performed by first modifying the database to accommodate citations. After that, almost all CIDS code was analyzed to integrate the citation concept in it. Then, new web-services were developed to enable the several CIDS components view and modify citation content. Also, due to the nature of CIDS - accessing Google Scholar to obtain the data it needs - the component responsible for contacting Google Scholar was modified so that citations could be obtained from Google Scholar;

Data verification This objective was not performed due to time constraints. The approach that would be taken to perform this objective would fall in, first, researching and then using tools that detect patterns on data provided. The data in this case would be user preferences on publications and citations (for instance, filtering and changing citation type). What would be expected in using these tools would be predicting user preferences based on past ones. Visually, this would be the user having automatically selected for him some publications and citations and others not;

Group Analysis This objective was performed by first finding the existing steps that formed a group analysis task. After that, a program was created to execute those steps sequentially, without requiring any user interaction. The interface was also modified to receiving input from the user, so that he could receive notification afterwards;

Security This component, albeit different from previous ones because it's nature is more supportive than functional, was implemented by using a PHP framework that already had security tools built-in to prevent many attacks web applications are target. The security component was focused on *User Profile* functionality, because it is where user interaction was more concentrated. The security measure taken, besides

using the framework mentioned that blocks common attacks such as *SQL Injection*, was basically using a *token* on the communication between user and server. This *token* prevented, to some extent, impersonation attacks by a malicious user;

Cross-Browser This objective, that also has a similar supportive nature like previous one, was performed by using a Javascript framework, *jQuery*, that abstracts Javascript native functions that only work on some browsers. By using *jQuery*'s functions, it is assured that those will work on all browsers, such as, *Firefox*, *Internet Explorer* and *Chrome*. Since previous CIDS code was not cross-browser compatible and was created using another Javascript framework *Prototype*, my first thought was try to convert it to *jQuery*. However, because there were functions that only existed in *Prototype* and not on *jQuery*, adding the fact that rewriting all existing Javascript code could take a very long time, I decided to only to use *jQuery* on the development of the new functionalities. This approach means that only the new functionalities implemented became cross-browser compatible;

1.4 Contributions

This work contributions are the following:

- An upgraded CIDS's website with new features;
- A repository of CIDS in Google Code, for future development.
- An walk-through on how to use CIDS.
- Evaluation of team LASIGE.

1.5 Document Structure

This document is structured as follows:

Chapter 2 Explains concepts that help understand the bibliometrics subject and also the state of art of bibliometrics tools. In the end, the work plan is shown and compared to the preliminary report;

Chapter 3 Shows and explains each component of CIDS 3.0 architecture;

Chapter 4 Describes in detail the work that I have done;

Chapter 5 Presents results of new CIDS working;

Chapter 6 Shows conclusions and future work;

This work was done entirely inside the facilities of *Faculty of Sciences of University of Lisbon*, for the research unit *LASIGE* (Large-Scale Informatics Systems Laboratory).

Chapter 2

Related Work

2.1 General concepts and state of the art

When a scholar or researcher needs background information (conclusions or results) to support his work, he or she usually includes a reference to that publication. This action is called *citing*, which is formally described as *the process of acknowledging or citing the author, year, title, and locus of publication (journal, book, or other) of a source used in a published work*[1]. If a publication gets mentioned several times, its number of citations increases and, probably, because made a positive impact on the scientific community and it would be useful if he could measure it. The metrics used for measuring citation impact are called *bibliometrics*.

The following subtopics explain the main concepts behind citation impact subject, and the state of the art in each one of them.

2.1.1 Bibliometrics

The word "metric" means a discriminatory number used to measure something. The number of citations, or publications, are two examples of the several metrics that can be used to assess the impact of an author/academic. Metrics used for this type of assessment - determining the quantitative value of the impact of one or more author - are called *bibliometrics*, or *informetrics.*

The simplest form for assessing the citation impact of an academic is by counting all of his citations and cited papers. This form of assessing impact is rather naive because it can not differentiate an academic senior with many papers and an academic with a "one hit wonder" paper[7] (that is, a paper that got highly cited).

To combine these two facets of publishing – *quality* (that is, the productivity or the number of published papers) and *quantity* (the impact, that is, the number of citations) - *Hirsch* introduced a metric, called *h-index*[8]. An academic only has high *h-index* if he has these two facets high. Formally, the h-index is defined as follows:

[1] http://en.wikipedia.org/wiki/Citation_impact

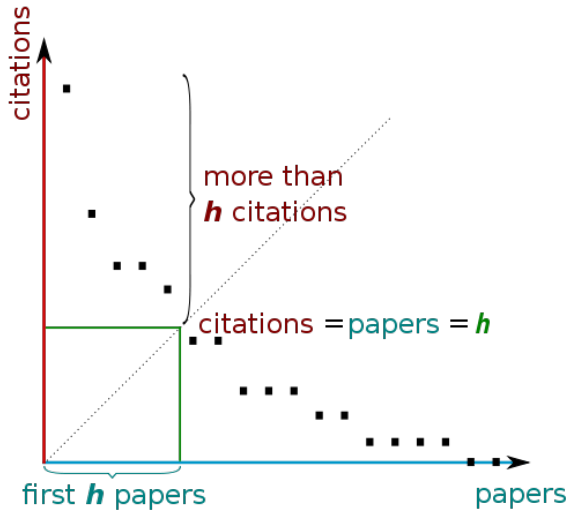


Figure 2.1: The h-index graphically. By plotting the number citations of papers in decreasing order, and then intersect that with the line with equation $y=x$, we can obtain the h-index.

A scientist has index h if h of his N_p papers have at least h citations, and the others (N_p - h) papers have no more than h citations each.

N_p is the total number of papers.

For instance, an academic with *h-index* equal to 20 means that he/she has 20 papers that have, at least, 20 citations each. The *h-index* can determined by plotting a graphic the publication citations counts (Fig. 2.1). The graph is created by simply plotting the number of citations in decreasing order and a line with equation $y=x$ that is, a line where the number of publications and citations are the same)[2]. The number of papers "before" (that is, in left of) the intersection of the lines mentioned before, is the *h-index*[13]. In Fig, 2.1, this value is five.

There are, however, some limitations[3] when using only *h-index* to assessing citations impact:

1. It is not appropriate for junior academic, due to the difficulty in increasing the *h-index* in yearly stages;

2. Once a paper belongs to the top *h* pages (the "left" side of the 45 degree line), its subsequent citations no longer count ;

3. It does not decline (academics that retire stay with the same *h-index*);

4. It does not take into account the number of authors on a paper (*co-authoring*);

[2]Or, if the axis are in same scale, a forty five degree line from the origin.
[3]http://www.harzing.com/pop_hindex.htm

To overcome the first limitation, a modified *h-index* was proposed, also by *Hirsch*, called *m*. This metric is the result of dividing *h* by the author's years of publishing, resulting in a more fairer comparison between junior and senior academics However, due to the nature of its calculus, *m* may vary greatly during the first years of an academic; it only stabilizes later on the academics career. Even so, it still is a useful alternative to *h-index* when comparing juniors and seniors academics.

To overcome the second limitation, a variant of *h-index* was proposed, the *g-index*, by *Leo Egghe*[4], defined as follows:

> *[Given a set of articles] ranked in decreasing order of the number of citations that they received, the g-index is the (unique) largest number such that the top g articles received (together) at least g^2 citations.*

Fig. 2.2 shows a comparison between h-index and g-index. Whereas the h-index uses the individual count of citations (red figures) on its calculus, g-index uses the accumulated citations count (green figures). The g-index in this figure is the number of symbols in left of *g-index reference line*, which is eleven (green diamonds), and the h-index is the number of red symbols in left of h-index reference line, which is six (red diamonds). Leo Egghe's says that *g-index inherits all h-index good properties, and, in addition, it better takes into account the citation scores of the top articles*. Although this is just an example, it can be seen that, for a set of citations, *g-index* is always either bigger than or equals to *h-index*, meaning it has bigger discriminatory power.

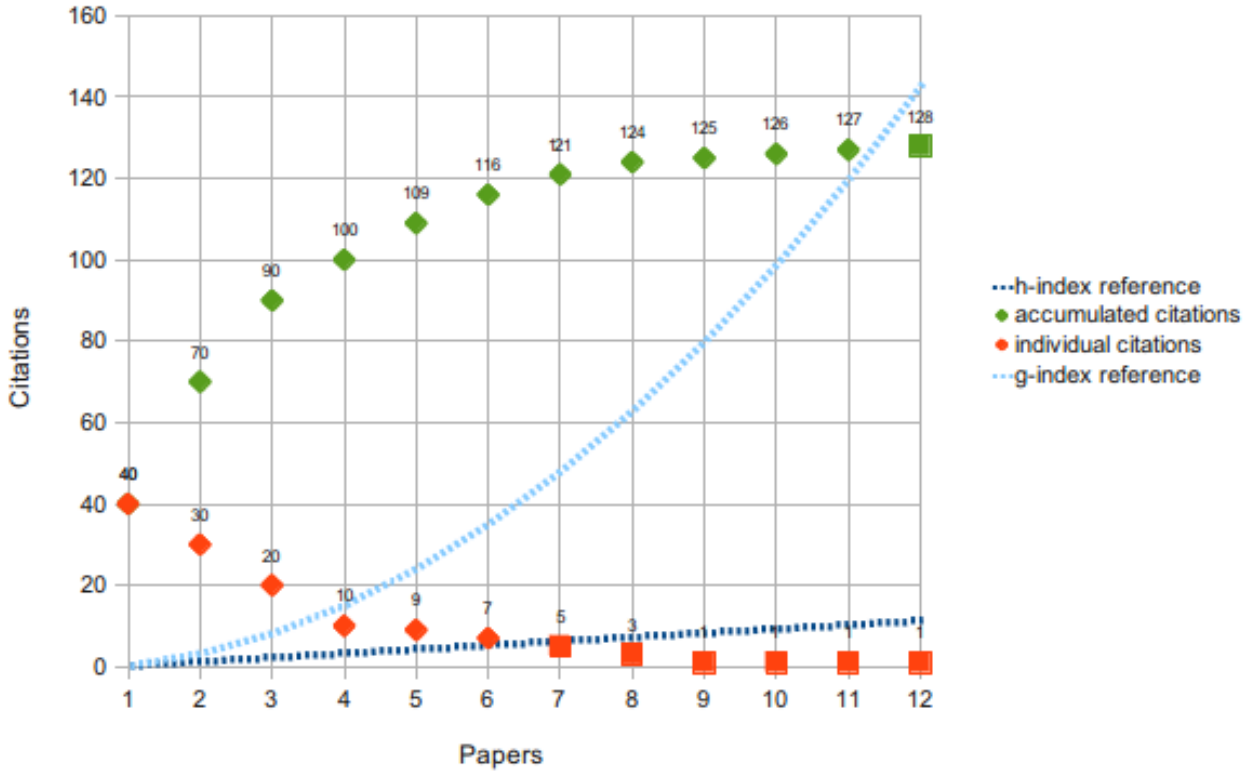


Figure 2.2: Comparison between h-index and g-index

In spite of not used much, *g-index* appears to be a great complement to *h-index*.

To overcome the third limitation, *Sidiropoulos*[4] suggested a *contemporary h-index*. This metric purpose is to add weight to paper's age: the more recent, the more impact it has, and the more older, the less impact it has. The consequence of this reasoning, is that, for juniors, their classic and contemporary h-indexes will be very similar, but for seniors, they can differ greatly. As such, when comparing *h-index* involving two age classes, both metrics should be used to complement each other.

Regarding the fourth limitation, Hirsch suggested partitioning citations among co-authors.

In spite of its limitations, h-index is still the most used bibliometric around citation impact subject. Nevertheless, in order to be used in decision making, we think it should be complemented with others, such as g-index, also the reason why we have it in CIDS.

2.1.2 Self-citations

Self-citation is usually defined as a citation where the citing and cited paper share at least one author [2] or co-author [14]. CIDS uses the definition of CiteSeer, where a citation is considered a self-citation if at least one of the author's names (in the citing paper) is located in the header of the cited paper. Depending on research area, the self-citation count percentage can make up between 10 to 30% of total citation count [1] [9].

There are reasons for an author to resort to self-citation [2]:

- To connect past and present work the of author and discussing previous findings without too much repetition;
- It is easier to cite a paper he knows well, which is also a reason for citing colleagues' papers of the same department, research group or university, instead of others;

But there are also less legitimate uses of self-citation [2]. For starters, it gratuitously increases one citations' count. Although neglected by Hirsch himself, some studies showed that the h-index decreases when we account only self-citations (from 12% to 1% [11][3]). This means that papers near h-index are obvious candidates for self-citing (for instance, the paper number seven in graphic 2.2, if added one more citation, it would increase the h-index). There is even an browser plug-in [5] that indicates how many citations are needed to increasing the h-index value by one.

Care should be taken when comparing authors from different areas of research due to the very nature of publishing: nature science academics tend to publish more (and smaller) papers/articles, while Social Sciences publish less (and bigger) papers/books[9].

[4]Generalized Hirsch h-index for disclosing latent facts in citation networks
[5]https://addons.mozilla.org/en-US/firefox/addon/scholar-h-index-calculator/

Some say that self-citation does not influence the quality of the publications[5] and it is a necessary action when there is no previous research in the area[6]. I think that, ultimately, it is better to have this discerning process than none at all: if we can have both data separately, we can compare it and, depending on the situation, draw conclusions.

As it will be mentioned below, currently, to the best of my knowledge, the only citation impact tool that does the task of discerning self-citations in its bibliometrics is CIDS. This discerning is done by advanced querying Google Scholar.

2.1.3 Publication collections

In order to have the most completeness publication and citation count possible in bibliometrics' results, it is necessary to have access to a large publication (and citation) collection, as well as easy methods to retrieve that information. Known publication collections include the following: *CiteSeerX* (former *CiteSeer*), *Institute for Scientific Information* (ISI) and *Google Scholar* (GS). Only the latter, which is used by CIDS, crawls the entire web (*CiteSeer* does not[6]) and does not have restrictions on its content (ISI has[7]). Both CiteSeerX and Google Scholar have public web interfaces that allows searching for publications and citations. Although, upon searching, CiteSeerX displays immediately the self-citations count, with Google Scholar, one has to perform advanced querying to be able to retrieve that information. There are also three downsides that we have found in Google Scholar:

- It does not have an API for easy and quick data retrieval;
- It temporarily blocks access when receives to many requests (from the same location) in a short period of time;
- Recent studies[10][12] showed that Google Scholar has *ghost authors*, that possibly influence the result of bibliometrics and because of that, it should not be used for citation impact analysis;

CIDS has managed to work around the three problems described. The first problem was resolved by parsing the HTML pages returned by Google Scholar and the second one by using *applet* technology, which shifts the origin of requests to the client. Regarding third problem, the disbelief in Google Scholar is due its crawler, which, firstly, invents author names and, secondly, inflates publication and citation count. According to those studies, the first situation (inventing authors) occurs when the crawler searches inside publications, for authors, titles and date information. Since the location and nomenclature of

[6] `http://citeseerx.ist.psu.edu/about/bot`

[7] `http://thomsonreuters.com/products_services/science/free/essays/journal_selection_process/`

Type of computation	Example of tools that use Google Scholar collection
On-line computation	• INRIA Lille[9] • QuadSearch[10] • CIDS
Script	• MATLAB[11] • *Harzing Publish or Perish*[12], also known as HPP, computes H and G index and other statistics. HPP service requires downloading a program, which is available to Windows and Linux formats. • Citations-Gadget[13]
Browser extension	• *Scholarometer*[14] is a cross-platform browser extension to help evaluate the impact of an author's publication. • *Scholar H-Index calculator*[15] is a lightweight Firefox extension which computes citation indexes for the submitted query on Google Scholar.

Table 2.1: List of citation impact tools categorized by type of computation

this information are not standardized, the crawler can inadvertently create artificial authors - the *ghost authors* - for those publications. The second situation (inflating publication and citation count) occurs when the same citation or publication is found in different places but counted separately, inflating both publication and citation count. Although this critics severely discredits Google Scholar fidelity, CIDS has some functionalities that can help alleviate this situation, such as the possibility to discard publications and, thanks to this work, also citations.

2.1.4 Citation Impact Tools

At the time of this writing, there are several tools that compute both h-index and g-index using Google Scholar as the publication data. These tools use different approaches to calculate those indexes, namely, using on-line computation, scripts or browser extensions[8]. Table 2.1 shows the tools categorized by the type of computation involved.

There are also other ways of computing *h-index* and *g-index* using other databases sim-

[8]http://en.wikipedia.org/wiki/H-index#Computing_the_h-index

ilar to Google Scholar. Some examples[16] are: Web of Science[17], which uses previously mentioned ISI publication collection, Scopus[18], Microsoft Academic[19] and Odysci[20]. This type of tools scrap the internet for publications and citations, each one having its own restrictions, saving the findings in their corresponding database, and present citation impact results. In terms of public access, all mentioned tools, except *Odysci*, are subscription based, that is, either its database or the journals/articles included are not open for public use. In terms of bibliometrics results, none of them discern self-citations.

It seems that Google is investing in citation analysis. Although it is in a limited launch, they have recently announced their new tool called Google Scholar Citations[21]. If they finish this tool and release it to the public, I believe CIDS will not be able to make a stand: they are the ones who hold the data. Their citation analysis would be faster, more reliable and with Google brand on them. In spite of this, CIDS still has the publication (and soon, citation) filtering functionality, as well as the group analysis, as leverage on this. Nevertheless, CIDS was a pioneer on citation analysis subject and it is always nice to see our work being acknowledged when big companies such as Google start to develop tools with similar behaviour like ours.

2.1.5 CIDS

As for CIDS v3.0, it is a tool that uses both on-line and off-line computation, but it does not require the user to install any special software. The only requirement is to have installed a valid Java Runtime Environment[22] (JRE) to have the capacity to run *applets*. Even this is an optional feature, if users do not want to use applets, no JRE is required on client side.

The difference between CIDS and the other tools, is that it calculates citations discerning self-citations, meaning that the results can sometimes be more accurate, as the number of citations and the h-index is sensitive to self-citations at the individual level[2], so if this can be in some way filtered from the results it can help improve accuracy.

CIDS, in its essence, is cross-platform because it only needs a browser with Javascript (JS) enable to work, meaning that CIDS also has more advantages when comparing to other tools that require the user to download software for these tools to work, or tools that only works on certain operating systems. Unfortunately, due to difficulties in performing Asynchronous *Javascript And XML* (AJAX) requests with a component used in CIDS

[16] `http://en.wikipedia.org/wiki/Academic_databases_and_search_engines`
[17] http://apps.isiknowledge.com/
[18] http://www.info.sciverse.com/scopus/
[19] http://academic.research.microsoft.com/
[20] `http://www.odysci.com/`
[21] http://googlescholar.blogspot.com/2011/07/google-scholar-citations.html
[22] http://www.java.com/en/download/faq/whatis_java.xml

Date	Milestone	Completed?
September 2010	Familiarization and evaluation with previous CIDS architecture and code	Yes
October 2010	Preliminary report	Yes
November 2010	Data verification	Yes
December 2010	Data validation	No
February 2011	User profile	Yes
April 2011	Assessment	Yes
July 2011	Demonstration of CIDS to LASIGE advisory board[23]	Yes
July 2011	Writing of thesis	Yes

Table 2.2: The milestones for this work

v3.0, the web browser *Internet Explorer* (IE) is no longer supported; a trait that CIDS v3.1 inherited.

When comparing CIDS with browser extensions to calculate indexes, we think that it is up to the users to choose if they like to use this kind of tools or if they prefer a more robust application, but even on this field CIDS has the advantage that not all browsers support extensions like the major browsers, limiting this type of tools to stay confined to a few browsers.

None of the tools mentioned above allows citation filtering, a functionality that CIDS v3.1 (this work) introduces. I believe that it will help authors filter incorrect citations (inherited from Google Scholar's database) from their publications, as well as adding missing ones, thus improving accuracy.

Finally, CIDS v3.1 has also the group analysis functionality, that enables the visualization of bibliometrics results for a group of authors. We think this feature is useful for comparing the impact of different research units, and perhaps promote competitiveness between them.

2.2 Planning

The planning for this work can be seen on Table 2.2.

The actual plan deviated from the original one because, in the first semester, I was not fully committed to this work due to two postponed course units from last year. As such, the functionalities planned for that semester took longer time to complete which inevitably pushed forward the ones that followed. Near the end, I arrived to a situation where I had to choose, between *user profile* and *data validation* (the most time-consuming features), which functionality was to be done using the available time. I decided to implement the feature *user profile*, due to its higher feasibility; the *data validation* feature required a

[23]This goal was set during the course of my work.

more thorough analysis on the *modus operandi* of the available tools (Weka[24] or Bow Toolkit[25]), followed by their respective integration with CIDS.

[24]http://www.cs.waikato.ac.nz/ml/weka/
[25]http://www.cs.cmu.edu/ mccallum/bow/

Chapter 3

Architecture

This chapter shows current CIDS architecture, by presenting its components and how they communicate with each other. In following chapters, each new functionality (mentioned in Section 1.2) will describe which and in what way this work has modified these components.

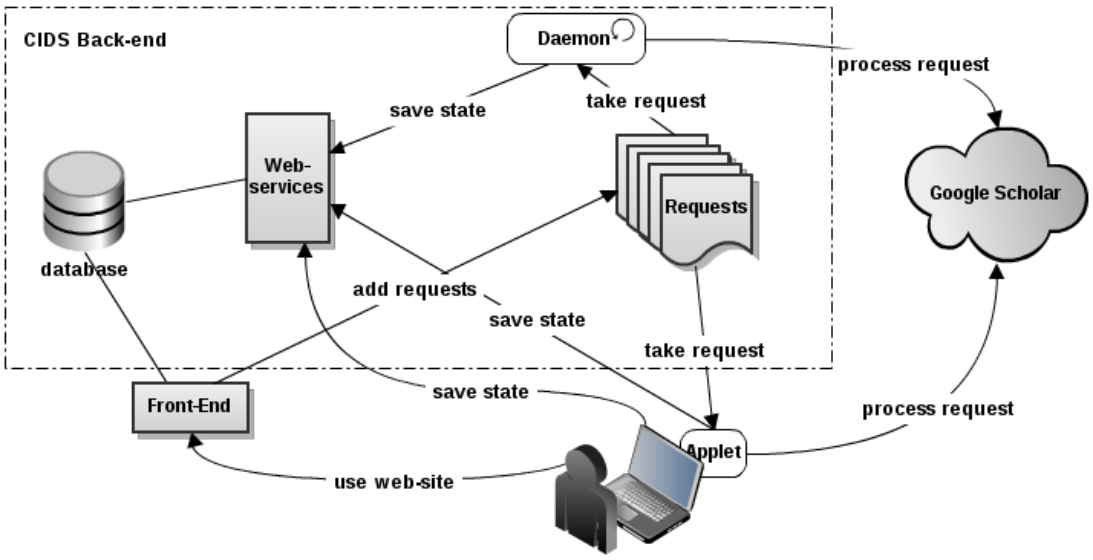


Figure 3.1: CIDS architecture and its interaction with external components

The main components that constitute CIDS (Fig. 3.1) are the following:

- Database
- Queue of requests
- Web-services
- Front-end
- Applet and Server (Daemon)

- Client

Each one of these components and to whom they interact, will be explained in the next sections.

3.1 Database

CIDS uses a database[1] to save data in a persistent manner. The database's schema[2] attempts to mimic the reality on how things relate to which other. In CIDS case, it tries to model how authors, queries, publications, citations connect to each other. CIDS current database schema can be seen on Fig.3.2.

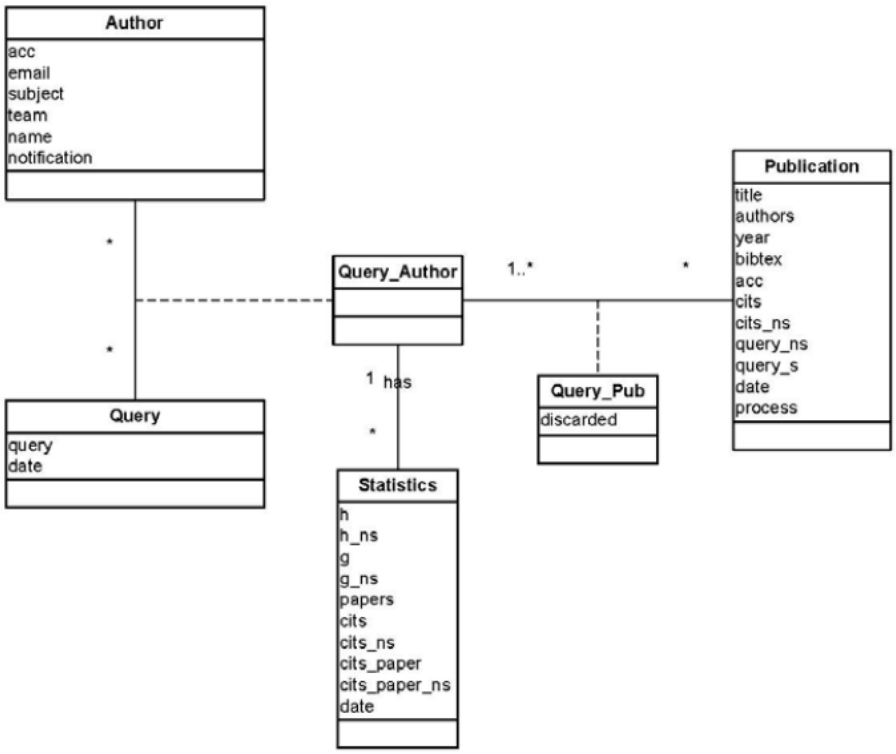


Figure 3.2: Database class diagram in UML

Each box in Fig. 3.2 represents a class. Each class can have properties, represented by field names inside each box, and relations/connections, represented by the lines that connect each box. Table 3.1 explains what each field of each table means.

For the sake of simplicity, the reference keys are not shown in database schema (Fig. 3.2). After analyzing Fig. 3.2, one can see that an author has many queries and a query can have many authors. This, however, it is not entirely true: first, a query only belongs to one author, second, though an author can have many queries, he only has one at a given

[1] organized collection of data
[2] Schema is similar to a blueprint of a house

Class	Table Field	Purpose
Author	acc	It means "accession number"and represents an unique identifier to each author/user of the system.
	email	The user email to where notifications are sent.
	subject	Options introduced by the author to help filtering the results obtained from Scholar.
	team	The team of the author, if he has one.
	name	The name of author on the team, if he has one.
	notification	Boolean field that allows the user to indicate the system weather he/she wants to receive email notifications or not.
Query	query	The search query that will be used to fetch citations from Google Scholar.
	date	The date when the query was performed.
Statistics	Bibliometric values: h, g, papers, cits, cits_paper	h: h-index ; g: g-index ; papers: number of papers; cits: number of citations ; cits_paper: number of citations per paper
	h_ns, g_ns, cits_ns, cits_paper_ns	The above bibliometrics, but discerning self-citation on their calculus.
Publication	title, authors, year	The title, authors and year of the publication.
	bibtex	A bibtex entry.
	acc	An unique identifier to each publication
	cits and cits_ns	Number of citations and self-citations of a publication.
	query and query_ns	The search queries used to fetch all citations and non-self citations.
	date	The date this publication was fetched
	process	The state of retrieval of this publication. Can have two values: *draft* or *full*. Publication with *draft* information is a publication without *bibtex*, *query_ns* and *query_s* parameters and/or without *cits_ns* parameter ; *full* is a publication with all parameters defined
	discarded	Stores whether a publication of a query was discarded or not.

Table 3.1: Database tables and fields.

time; the others queries are kept for historical reasons. What one should understand and conclude from this relation is that, a query belongs to only one author, and an author only has, at a given time, one query.

The association of a query and an author, which is stored in Query_Author, can have many publications, and each one of these can belong to many queries. This is easy to understand, since two queries can generate two groups of publications that can intersect each other. One can also see that the connection class, Query_Pub, has the *discarded* property. This property is used to know if some user has discarded some publication in some query. Due to publications being shared among queries, it is not feasible to have the *discarded* property related only to the publication. If that would be the case, some user preferences would be overwritten by others.

We can also see that Query_Author can have many statistics, which are calculated over time (hence the *date* field in *Statistics* table).

3.2 Queue of Requests

Since Google Scholar temporarily blocks access to a machine after receiving too many requests in a short period of time from the same location, CIDS required a structure that could schedule its requests to Google Scholar. The structure that was crafted to overcome this problem was a global queue of requests, used by both Server Daemon and Applet (more on this in Sec. 3.5).

The requests are added to the queue directly from CIDS front-end (the website) or when other components need it, via web-services. These requests basically encapsulate direct calls to Google Scholar. This encapsulation allows CIDS to schedule its requests, making more difficult for Google Scholar to block them.

The requests of the queue are saved to a database table named *daemon_stack*. This table is composed of the following fields:

Table Name	Table Field	Purpose
daemon_stack	author_acc	The author/user who started the request.
	pub_acc	The acc of the publication that this request is about.
	uri	The URL of the request, base64 encoded [3].
	uri_type	Requests of the queue can have one of the following types: **scholar_raw:** fetches temporarily information of the publications of a search query; **bibtex:** fetches the bibtex of a specific publication; **cits_ns:** gets the number of non-self citations of a publication;
	html_raw	The HTML fetched from Google Scholar, after accessing the *uri*. It is stored in format base64 encoding.
	state	Requests of the queue can have one of the following states, generally have them sequentially: **standby**: requests are normally added to the queue with this state. They just stay there, waiting to be used; **pending**: This state means that the requests are on the line to processed but have not started yet; **processing**: If a state has this state, it is means that it is being processed, that is, its *uri* is being accessed and the resulting HTML saved in *html_raw* field; **ready**: the request is completed/finished;

CIDS intervals requests with a 40-60 seconds delay. This is a empiric tested duration number which ensures that CIDS can make requests to Google Scholar without being blocked[4].

3.3 Web-services

Since CIDS is composed of components that were created using different technologies (PHP, Javascript, Java) and each one needs to access or alter CIDS state, that is, use the database, web-services were created to centralize and mediate this access. A web-service is a software technology that enables different applications to send and receive data in XML (extensible markup language)[5]. This allows different software systems to communicate with each other, regardless of the technology used. CIDS uses web-services to mainly unify operations to be done to the database.

A web-service is triggered when accessed by its respective URI – the location of web-service on the internet. By sending a request[6] to that URI, it activates it and then some

[4]During the time I was working on CIDS, it kept being constantly blocked by Google Scholar. I can only presume that Google Scholar has decrease the rate of the number of request per minute before restricting automated accesses to their resources.

[5]`http://en.wikipedia.org/wiki/Web_service`

[6]By "sending a request", I mean when someone accesses the URI on the browser (or accesses the port 80 of URI web-server.

operation is performed. To add more functionality to the web-service, it typically receives a list of parameters, whose format, like the URI, is known in advanced.

Table 3.2: Small list of web-services

Category	WS Example	WS Description	WS Parameters	Opt.
Author	add_author	Adds an author to the system and returns a unique identifier. If the author already exists on the system, the registered identifier is returned; otherwise a new one is created.	email	No
			query	No
			subject	No
			name	Yes
			notification	Yes
			format	Yes
Queue Of Requests	duplicate_daemon	Duplicates the information from a daemon_stack. This web-service is used when a URL already exists registered for an author and another author request the same resource. For instance, co-authors will have at least some publications in common, so when using CIDS system instead of making requests for every publication of each author and co-authors, the system will optimize the number of resource requests, be reusing information for different authors. When using this web-service, if the information being copied does not exist, then nothing happens	src_author_acc	No
			dst_author_acc	No
			uri	No
			format	Yes
Publication	get_pub	Gets a single publication given its identifier (acc). If no publication exists with the given acc, an error message is returned instead.	acc	No
			format	Yes
Statistics	get_stats	Calculates and updates the statistics for an author, given his identifier (acc)	acc	No
			format	Yes
Support	send_email	Works as the mail server for the entire application. All components of the system that needs to send email use this web-service.	subject	No
			from	No
			to	No

			msg	No
			format	Yes

Web-services can be used in several ways ([7]):

RPC Invokes operations on remote objects;

Service oriented Instead of operations, like RPC, its main unit is messages, which are sent over the network, commonly in XML. These messages are sent to existing services (like printing service) which then respond accordingly;

RESTful Attempts to transfer state of entities by using the HTTP protocol, using it's methods: GET for fetching information, POST for posting information, PUT for updating information and DEL to delete information);

The web-services present in CIDS are an hybrid of "Service Oriented" and "RESTful" approaches. They transfer state of objects using GET and POST methods of HTTP (the REST aspect) and use messages in XML to perform the transfer of state of entities, which is the "Service Oriented" aspect).

Given the many web-services CIDS has, I have categorized them into groups, according to the object/entity they operate on. This categorization can be seen on Table 3.2. For each category, an example of corresponding web-service is given, followed by its description and the parameters it receives and if these are optional or not.

CIDS web-services' URL's are constructed as follows. The start of the URL is with "http://" followed by SERVER_URL, whose value depends of the name given to CIDS website. Then, it is concatenated with "/api/?", followed by the name of the web-service and respective parameters, using "&" for separating them and "=" to assign values to them. For instance, if one were to access the URL:

```
http://SERVER_URL/api/?add_author&email=email@server.domain
&query=author:my query "Lisbon or Lisboa"
-author:remove-name&subject=eng or soc&format=xml
```

, it means that he was accessing the *add_author* web-service with the following parameters:

email: email@server.domain

query: author:my query "Lisbon or Lisboa" -author:remove-name

subject: eng or soc

[7] http://en.wikipedia.org/wiki/Web_service

format: xml

Every web-service returns something. CIDS web-services returns a XML string, if "format" parameter is provided with value "xml". In previous example, CIDS would return something similar to:

```
<?xml version="1.0"?>
<cids_api>
  <add_author>
    <result>OK</result>
    <type>INSERT</type>
    <acc>4309230507110118514 6</acc>
  </add_author>
</cids_api>
```

The rest of web-services are used in a similar manner, differentiating only in the name of the web-service and its parameters.

Since the *front-end* engine (discussed in Section 3.4) resides on server-side, it can access the database directly, without using web-services. This may seem a violation in CIDS architecture, because I previously mentioned that the web-services' purpose was to unify the access to the database and now I am stating that front-end does not use them. The reason for this to happen is due to the same location and technology that front-end engine and web-services share: both reside in server-side and are written in PHP. The only difference in them is what they output: front-end engine outputs user interfaces and a web-service outputs XML strings (similar to example mentioned before). Plus, if front-end engine would use web-services to access database, it would be rather inefficient. An analogy of this inefficiency would be having two brothers living in the same house corresponding with each other using the local post office, rather than just talking with each other.

The rest of CIDS components, however, need to use web-services.

The Daemon and Applet components (presented in Sec. 3.5) mainly use web-services that are related about changing the requests within the Queue of Requests, namely, requests' state and content.

The client (through AJAX[8] calls) mainly use web-services to change publication discarding state.

The rest of the web-services are used by programs written in JAVA language that are invoked during user interaction with CIDS.

It is usually named Web API (Application Programming Interface) to a group of web-services that allows the access and alteration of a system state, without revealing its implementation. In CIDS case, the web-services are for internal use only, though anyone could use them as they see fit.

[8] AJAX calls are made within user browser while he is using the website.

3.4 Front-end

The front-end is the interface between the user the rest of the system (back-end). The front-end is, typically, a set of web-pages, usually created using markup language, such as HTML, and stylized with CSS. Additionally, Javascript[9] can be used to add more interactivity. The front-end engine is the application logic that outputs those interfaces.

CIDS it essentially composed of three interfaces/pages, created using these programming languages. The first interface is about providing the query search, the second about publication filtering and the last one about presenting the results. Only in the first two (Fig. 3.3 and Fig. 3.4) pages it receives input from the user.

To understand how these interfaces are related, I am going to provide a walk-through on how to use CIDS.

3.4.1 Query Creation

When accessing CIDS, the user is presented with the screen of similar to Fig. 3.3

In the top section on this screen, there is a form with fields *email* and *query*. The first is used for notification purposes and the second is for inserting a search query of an author, using Google Scholar format[10].

[9] Javascript is a scripting language that resides on the client-side, inside the web page and typically adds more interactivity on user experience. It should not be confused with Java.

[10] Search queries tips can be seen here: `http://scholar.google.com/intl/en/scholar/refinesearch.html`

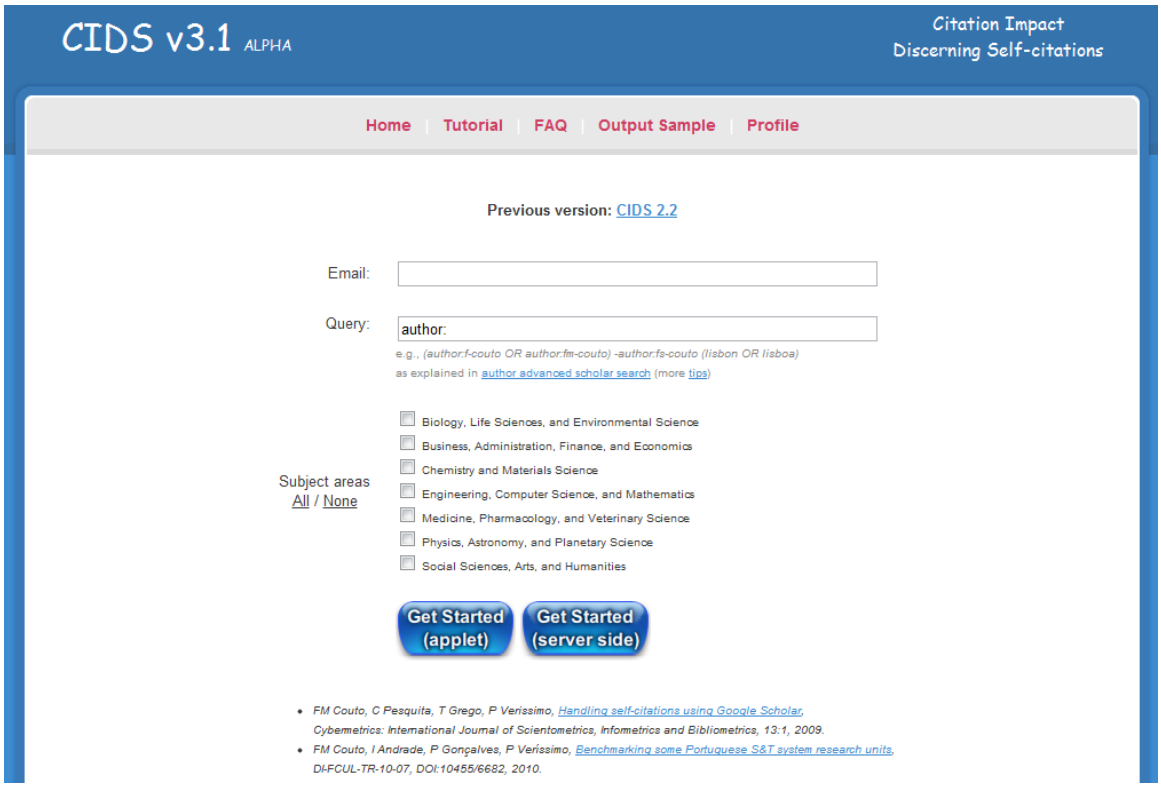


Figure 3.3: CIDS homepage

In the middle section of this page, user can specify which of the following subject areas of interest he wants regarding previous search query:

- Biology, Life Sciences, and Environmental Science
- Business, Administration, Finance, and Economics
- Chemistry and Materials Science
- Engineering, Computer Science, and Mathematics
- Medicine, Pharmacology, and Veterinary Science
- Physics, Astronomy, and Planetary Science
- Social Sciences, Arts, and Humanities

There are also two shortcuts that enable the user to either select or discard all areas subjects in one click.

In the bottom on the screen, there are two buttons. In order to get started, user must click on *Server* or *Applet* button. *Server* queues user's requests in a global queue, common to every user. If the user chooses this method, he will have to wait more time to have his request processed. If, instead, he chooses *Applet*, that waiting period is reduced (this is explained in Section 3.5). Whichever method chosen, both do the same thing: fetch publications from Google Scholar.

3.4.2 Publication Filtering

After pressing one the buttons mentioned, some status messages will show up and then, if everything works normally, a screen similar to Fig. 3.4 is presented.

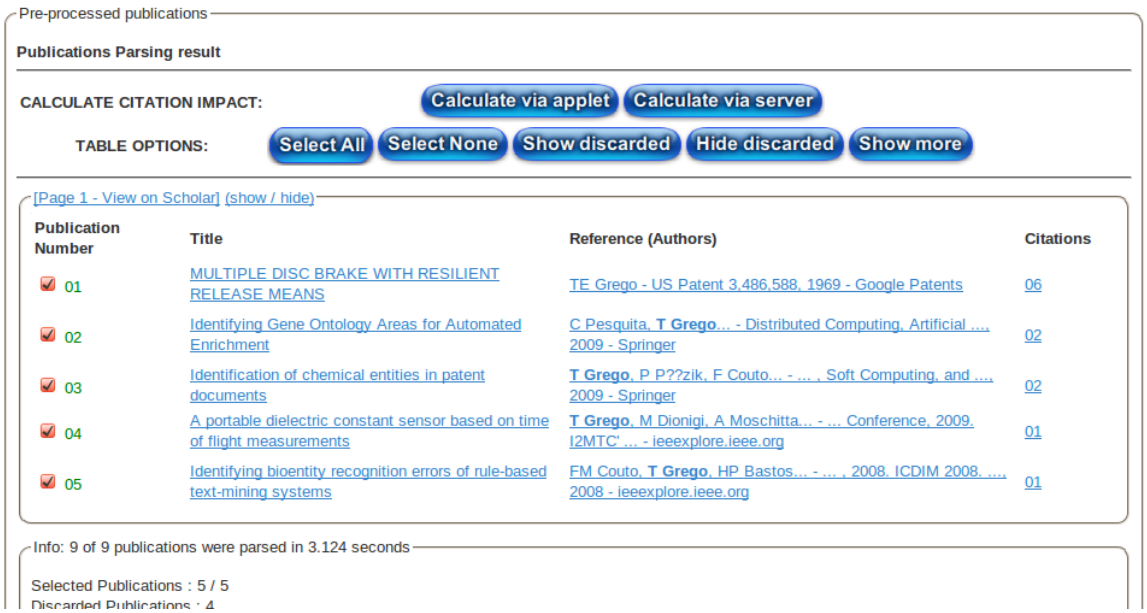


Figure 3.4: Publication Filtering

This interface (Fig. 3.4) is constituted primarily by a table of publications, which are related to previous search query. These were fetched from Google scholar in a first sweep. The information presented for each publication is the following:

number: The order which appeared in Google scholar;

title: The title of the publication;

reference (authors): The authors of the publication. This text is only a draft because it is the result of parsing search results. Later, CIDS is going to correct it by parsing the specific publication's bibtex;

citations: The total number of citations;

If user encounters a wrongly fetched publication, in this interface he has the opportunity to discard it. This action enables the user to select which publications he wants to be included during final calculus. It is also possible to select or discard all publications at once. After this, the user has again to choose between *Server* or *Applet* (Section 3.5). But, in this case, if he chooses *Applet*, then he will have the opportunity to contribute to citations calculus, by, in conjunction with CIDS server, resolving requests to Google Scholar. Should he chooses *Server*, he must wait for his requests to be resolved by CIDS server, just like any other user.

3.4.3 Bibliometrics Result

The final interface is divided in three sections (Figure 3.5, Figure 3.6, and Figure 3.7). If citations calculus is in progress, shown by a progress bar between the second and third sections, all results shown in each section are not definitive. The entire page reloads itself from time to time, until calculus is complete.

On top of CIDS results (Figure 3.5), bibliometrics of the author, accounting only publications previously chosen, are presented:

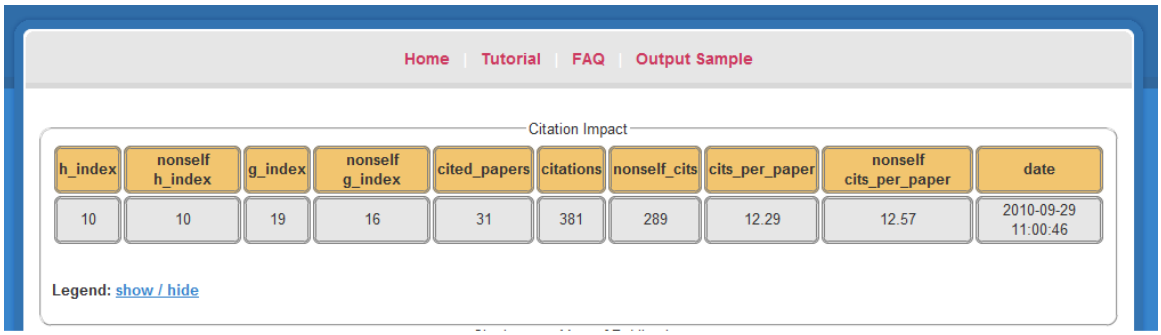


Figure 3.5: CIDS bibliometrics

- h-index (with and without accounting for self-citations) ;
- g-index (with and without accounting for self-citations) ;
- number of cited papers ;
- number of citations (with and without accounting for self-citations) ;
- number of citations per paper (with and without accounting for self-citations).

In the middle of the page, there is a graph (Figure 3.6) that shows the number of citations an author had over the years, discriminating self-citations in each one. The bars indicate the number of citations on the respective year; the darker blue only accounts for non-self citations, while the other one accounts for all citations.

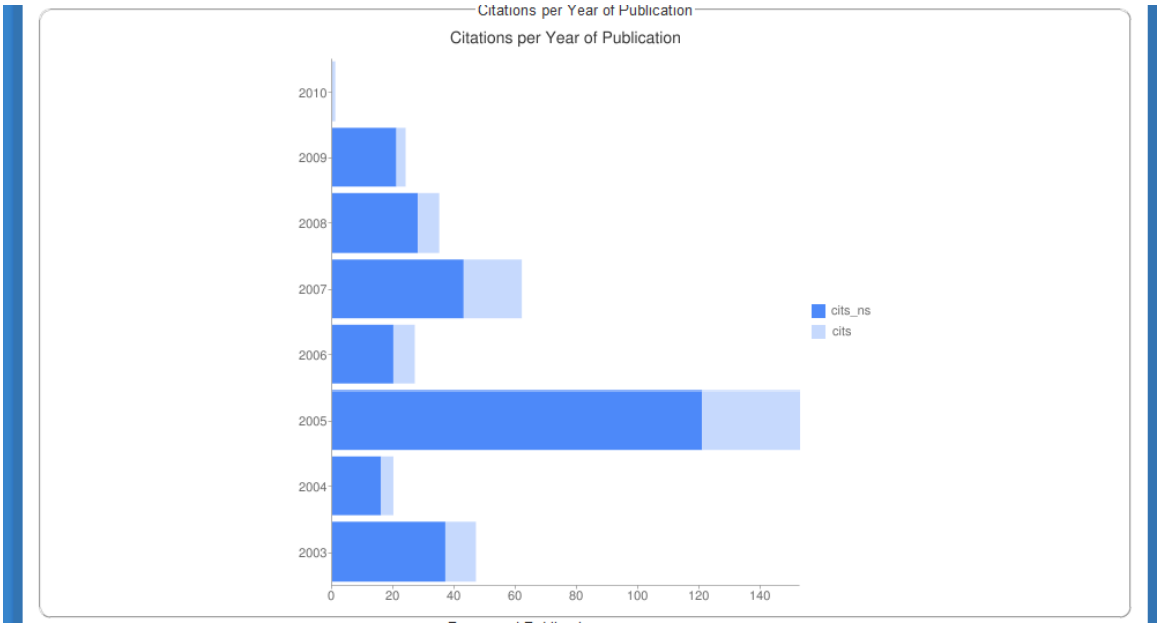


Figure 3.6: CIDS graphic bar

At the bottom of CIDS results (Figure 3.7) there is a table, with the following fields:

Processed Publications READY 31 OF 31

Title	Authors	Year	Citations	Nonself	Self
Facts from text-is text mining ready to deliver	Rebholz-Schuhmann D and Kirsch H and Couto F	2005	85	70	15
Measuring semantic similarity between Gene Ontology terms	Couto FM and Silva MJ and Coutinho PM	2007	38	31	07
Metrics for GO based protein semantic similarity: a systematic evaluation	Pesquita C and Faria D and Bastos H and Ferreira A and Falcao A and Couto F	2008	35	28	07
Implementation of a functional semantic similarity measure between gene-products	Couto FM and Silva MJ and Coutinho PM	2003	34	28	06
Semantic similarity over the gene ontology: Family correlation and selecting disjunctive ancestors	Couto FM and Silva MJ and Coutinho PM	2005	34	26	08
Finding genomic ontology terms in text using evidence content	Couto F and Silva M and Coutinho P	2005	26	18	08
GOAnnotator: linking protein GO annotations to evidence text	Couto FM and Silva MJ and Lee V and Dimmer E and Camon E and Apweiler R and Kirsch H and Rebholz-Schuhmann D	2006	23	18	05
Semantic similarity in biomedical ontologies	Pesquita C and Faria D and Falcao AO and Lord P and Couto FM	2009	20	19	01
Classifying biological articles using web resources	Couto FM and Martins B and Silva MJ	2004	15	11	04
Evaluating go-based semantic similarity measures	Pesquita C and Faria D and Bastos H and Falcao A and Couto F	2007	14	10	04
ProFAL: Protein functional annotation through literature	Couto FM and Silva MJ and Coutinho P	2003	07	05	02
Proteinon: A web tool for protein semantic similarity	Faria D and Pesquita C and Couto FM and Falcao A	2007	06	00	06
Improving information extraction through biological correlation	Couto FM and Silva MJ and Coutinho P	2003	06	04	02
Figo: Finding go terms in unstructured text	Couto FM and Silva MJ and Coutinho P	2004	05	05	00
Finding genomic ontology terms in unstructured text	Couto F and Silva M and Coutinho P	2005	04	04	00
Measuring semantic similarity between gene ontology terms. DKE-Data and Knowledge Engineering	Couto F and Silva M and Coutinho P	2006	04	02	02

Figure 3.7: CIDS publications result

title: Publication title;

authors: The final version of the authors of the publication;

year: The year the publication was published;

citations: Total number of citations of publication;

nonself: Number of non-self citations;

self: Number of self citations;

If publications calculus is not yet completed, it will be reflected on the background of publication's title. If it is shown in red (Figure 3.8), it means that no requests to Google Scholar (of that publication) has been resolved.

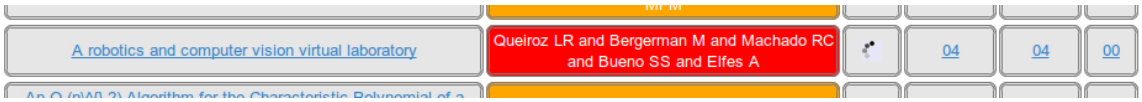


Figure 3.8: Publication in red - no resources have been processed

If yellow (Figure 3.9) , then some requests have been resolved, but there are still some left. Finally, if gray (Figure 3.7), then all publication's requests have been resolved.

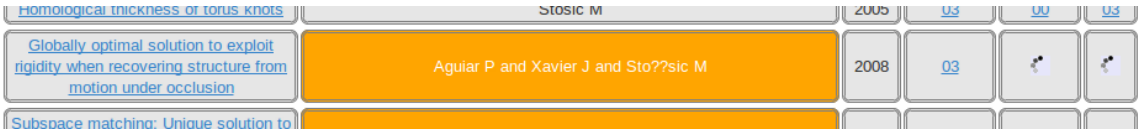


Figure 3.9: Publication in yellow - some resources have been processed

If user has chosen "APPLET" in previous page (Fig. 3.4), then a section similar to Fig. 3.10 appears on the top of results page.

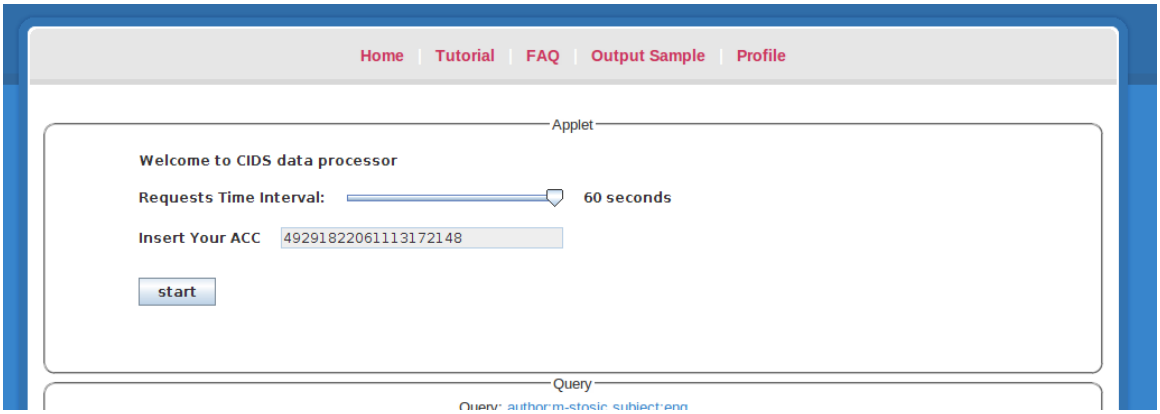


Figure 3.10: Applet for aiding in calculus

The applet - a software program embedded on the page - mimics server behavior by resolving requests, but here, the source of the requests is the user's machine, not CIDS (more on this in section 3.5).

Most of the web-services that exist are used during this step. When passing from Fig. 2 to 6, a different web-service is invoked. This web-service invokes a program written in Java, which, in turn, invokes more web-services. It is in these Java programs that most of the web-services are used. In analogous manner, the passing from Fig. 6 To Fig. 4 also invokes another program in Java which in turn invokes other web-services.

3.4.4 Group Analysis

There is another interface whose access is both hidden and restricted (by password) to the regular user (Fig. 3.11). This interface allows the administrator to control and accompany the cycle of requests. For instance, it can change the state of a request which, for some reason, got stuck in "processing" state, to "pending" or "ready". It also allows the administrator to calculate bibliometrics for a group of authors. He just needs to upload a correctly formatted comma-separated-value (.CSV) file. For instance, the following text, saved in a file with extension .CSV is considered correct:

```
Email,Query,Subject,Team,Name,Notification
the_one@gmail.com,author:ivan ricardo„Andrade,Ivan,no
ivan@lasige.di.fc.ul.pt,author:josé vieira andrade„Andrade,Ivan
Andrade,no
```

Here are the rules that the file must obey:

1. Use double-quotes instead of single-quotes;
2. Use comma (",") to separate values;
3. These fields are mandatory: email, query, team and notification;
4. The remaining fields, Subject and Name, should stay in blank, if they are not needed;

The current *modus operandi* for starting bibliometrics calculus for a group of authors requires many manual steps. After uploading the comma-separated-value file, one must must wait for each "first_req" request (of each member) to finish. After that, one has to search for the team's name in the "ready" pile of requests to retrieve those "first_req" requests. Then, one has to select them and click on a button so that the system creates the publication requests of each individual member. Finally, in the "standby" pile of requests, for each member, one has to search for its requests and change their state to from "standby" to""pending", for later processing.

This functionality will be revised (section "team analysis") and it will become more automated: it will only need the step of file uploading.

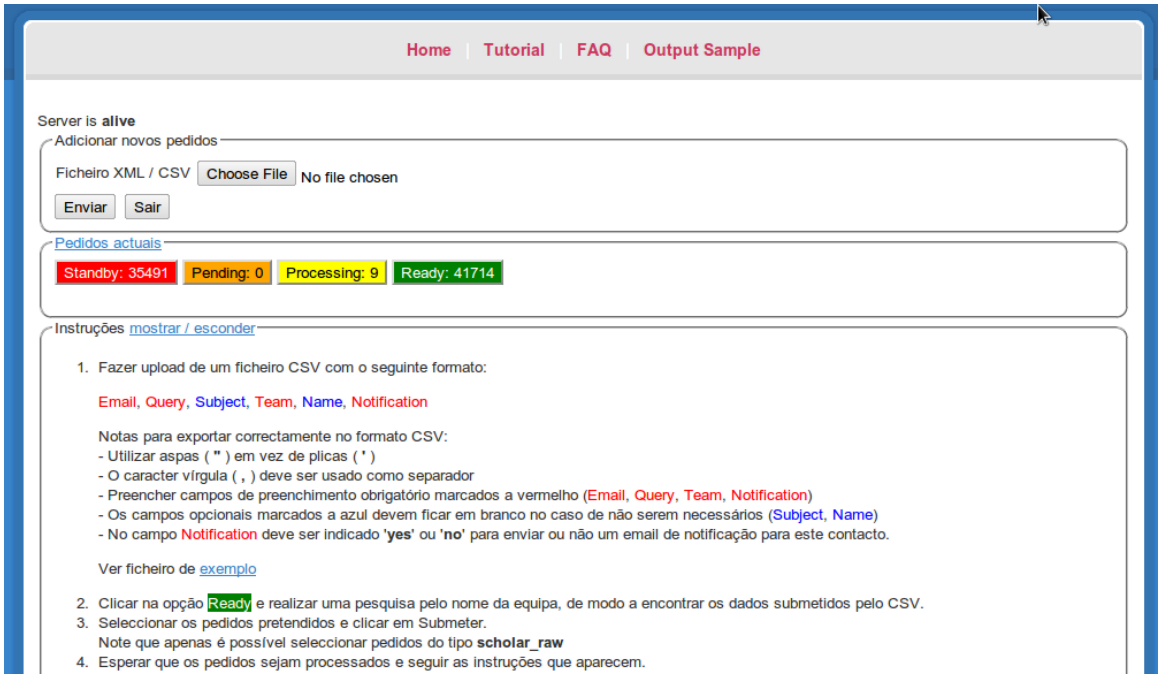


Figure 3.11: CIDS administration interface

3.5 Applet and Server (Daemon)

There are two components that have very similar behavior but need to be in distinct locations on CIDS architecture: the applet and the server daemon. The reason for this will be explained next.

By definition, a *daemon* is a computer program that runs in the background, rather than direct control of the user, and it is usually initiated during the operating system start-up.[11] CIDS has a program with similar characteristics - runs in the background - reason why it was named *daemon*. This daemon is located on server-side and its main responsibility is to fetch requests with pending state from the queue of requests, and process them, until there are none. It has the additional responsibility of sending notification email to a user after all his requests are finished.

An applet[12] is a small application program that is included in web pages that are display on user's browser. CIDS' applet shares daemon's main responsibility.

The reason for having a program with similar behaviour of daemon's in client-side, is due to a restriction measure of Google Scholar's that prevents automated access to its resources. As mentioned before, when Google Scholar detects a machine performing multiple requests, it temporarily blocks its access. This is why CIDS applet needs to have daemon's behavior on client-side: since requests origin are done from a different machine

[11] http://en.wikipedia.org/wiki/Daemon_(computing)

[12] When “applet” is mentioned, I am referring to the Java applet, that is, a program written in Java that is included in a web page: `http://en.wikipedia.org/wiki/Java_applet`

- technically, it is from a different IP[13] - it virtually increases the rate of requests, per period of time, CIDS can do.

One important note to consider is that CIDS only uses the applet on a user's browser to work on requests initiated by him, not by others.

Both daemon and applet are programmed in Java language.

3.6 Client Browser

Besides enabling applet usage, the client's browser also has another important role in CIDS. Through AJAX calls[14], it is possible for the browser to communicate with server without needing to refresh the page. This makes possible for the user to select/discard publications (Fig. 3.4)) and have their state immediately[15] saved on the server's database. This results on a more seamless experience for the user. There are others scenarios where AJAX is used: automatic publications state update(Fig. 3.7), bibliometrics (Fig. 3.5) and graphic bar (Fig. 3.6), during final calculus.

[13] IP (Internet Protocol) is the public address of a computer on the internet.

[14] AJAX (asynchronous Javascript and XML) calls allows browser to send requests to server for processing, and receive response, without refreshing its current page.

[15] Depending on network latency.

Chapter 4

Work Done

This chapter is divided in several sections, each one describing the work done on each new functionality described in Section 1.2. For each new functionality, I present the architectural components, shown in Section 3, that required modification (see Table 4.1 for an overview) and also how and why I have accomplished that. I have also added to this chapter the work done on auxiliary functionalities (functionalities with a supportive role) (Section 4.4) and also some unplanned work (Section 4.5).

Functionality	Database	Front-End	Queue of Requests	Applet & Server	Web-Services	Browser
User Profile	x	x				
Data Verification	x	x	x	x	x	x
Group Analysis		x		x		

Table 4.1: Matrix showing the new functionalities and with which CIDS architectural components their development coped with

4.1 User Profile

User profile is a restricted area where a user can visualize all its search queries, as well as keep track of their progress. In order to a new user to access this area, he first has to complete two steps: register and authenticate on the website. If the user is already registered, he only has to authenticate. Should he succeed, he enters in his restricted area. If the user does not remember his password, he can perform a password reset, and a new password is sent to his email.

The objective is to give to users the possibility to administer multiples queries simultaneously. The problem with this is that it implies changing a core property of CIDS: rather one query, having many queries per author. Like has been mentioned in Section 3.1, it is on the *database* component where the relationship between authors and queries is defined. As such, this component was the first one to be analyzed. Furthermore, new interfaces are

also required to interact with user, for instance to present all his queries, meaning that the *front-end* component, which is responsible for creating those interfaces, was also subject of review. Both components are discussed next.

4.1.1 Rationale on Database

First Approach

My first approach involved modifying in how CIDS internally operated. In order to show some arbitrary bibliometrics results, the system needs only an identification of the author, because it can infer his query. But it is this act of inferring that complicates the implementation of this functionality; it binds author and query and there is no way of outputting the bibliometrics result of an arbitrary user, other than his latest one. This problem is illustrated by the following pseudo-code:

Require: $\$author\_id$

1: $\$query\_id \leftarrow get\_most\_recent\_query\_of\_author\ (\$author\_id)$
2: $\$results \leftarrow get\_bibliometric\_results\_from\_query\ (\$query\_id)$
3: $print\_results(\$results)$

This pseudo-code purpose is to output bibliometrics results of an author. We can see that, first, it receives the identification of the author (line 0), it then gets author's most recent query (line 1), then gets the results that query (line 2) and finally displays them (line 3). The problem evidenced before is evidenced on lines numbered 0 and 1. Ideally, we would like to have something similar to the next pseudo-code, where, instead of passing the identification of the author, we pass the identification of the query, thus not binding author and query:

Require: $\$query\_id$

1: $\$results \leftarrow get\_bibliometric\_results\_from\_query\ (\$query\_id)$
2: $print\_results(\$results)$

In order to do this, it would be necessary to first, find all occurrences where *$query_id* is inferred by *$author_id* and, second, try to inject *$query_id* into the same context, differently. To help pinpoint which functions, on CIDS' code base, had this problem, I developed a map of dependencies of functions. This map was conceived by performing a string search all over CIDS' source code, containing either *query_id* or *q_id*, then I searched on the functions that mentioned previous ones, and so on, until there were no more to reference. My findings can be seen on Appendix B and resemble a horizontal tree, where functions in scripts are bound to other ones on others scripts.

The visualization of the flux of information of illustration in Appendix B made me perceive that, when *Server Daemon* (Section 3.5) is processing a request from the queue of requests, it does not (and it cannot) know to which query the request is related to, because it only has knowledge (besides it's specific information such as its type) of the

author who created it and its target publication. This first "clue" cautioned me that current approach I was considering would probably not work on enabling multiple queries per author.

But what really put an end to this approach was something I discovered after creating the mentioned map in Appendix B, although these events are not related; given the current database structure in Figure 3.2, if I were to connect *Author* to *Query*, all author's new queries inherit same subject because the *subject* property belongs to the *Author*, not *Query* class. This means that all the queries of one author would always have the same subject, something that is definitely not desirable. It also means that, even I could solve previous problem, authors still could not have multiples queries with different subjects, simultaneously. This approach tried to modify something that was *core* on CIDS, which revealed a failure and has cast me into trying to look at the problem from another perspective, which I describe next.

Second Approach

This second approach involves in only extending how CIDS works, as opposed to modifying it, like previous one.

Since one author has only one query (and one query belongs to only one author), one can use the terms "author" and "query" interchangeably. So, if a new class was to be created, called *user*, and then associated with author - meaning "user has many authors" - effectively, it would be the same thing as saying: "a user has many queries", which is precisely what is wanted. To accomplish this, I just needed to map users to authors. This mapping can be seen on Figure 4.1.

It makes more sense to me linking *user* to *author* because then I can say "a user has many authors, each one with a query". However, for all fairness, linking *user* to *query* would have the exact same result, because, as previously stated, the terms "query" and "author" can be used interchangeably.

One might notice that I am creating a linking class, *UserToProfile*, on the one-to-many connection between *user* and *author*, when such class is commonly on many-to-many connections type. This happens because on a one-to-many connection type, the "many" side needs to store a reference to the "one" side. In this situation, it means modifying the structure of a existing class, *author*. Because existing scripts are coded based on a specific *author* class structure, this is definitely not desirable. So, instead of altering existing data structure, I have merely extended it.

This approach is much more feasible than previous one because it does not modify existing code and, more importantly, each query has its own subject. All that was needed to be done was to add two new classes - *User* and *UserToAuthor* - and link them to previous database class diagram (Figure 3.2). The resulting UML class diagram can be

seen on Figure 4.1. The fields inside each class are explained in next section.

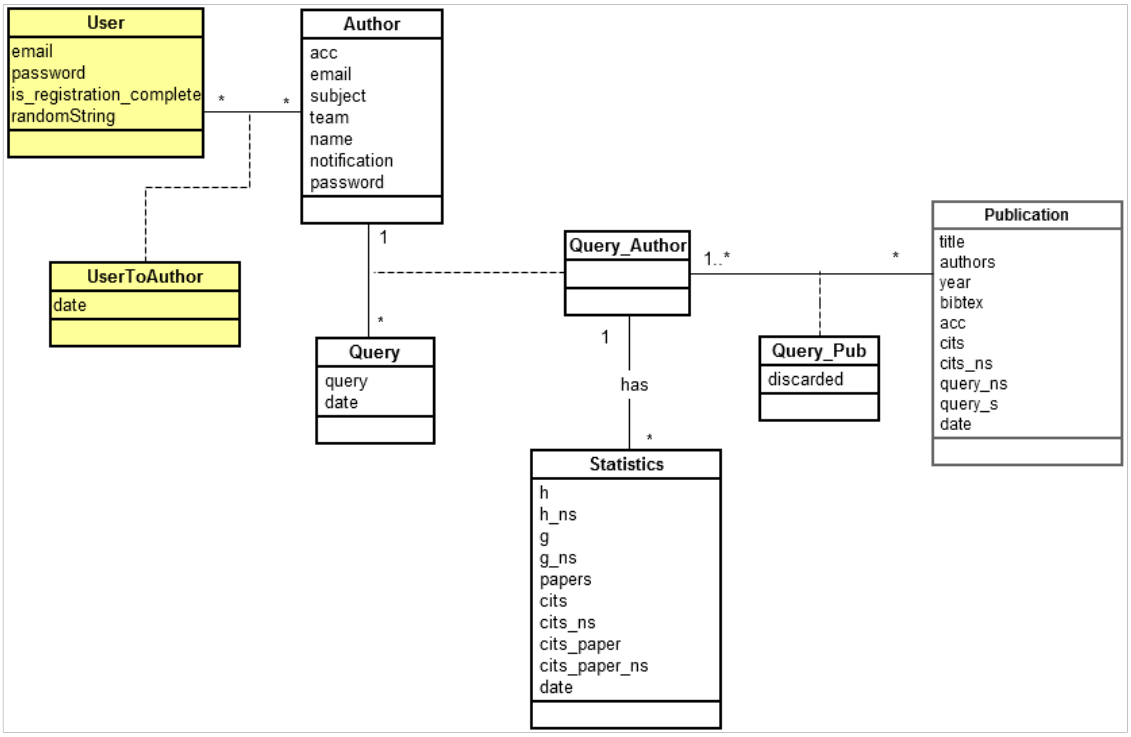


Figure 4.1: Updated UML class diagram with the new classes highlighted.

By modifying the database component CIDS can store multiple queries per author/user. Next component is *Front-End* which is the component responsible for interacting with user.

4.1.2 Front-End

Like has been stated previously in the beginning of this section, this functionality required new interfaces because it needs to capture new user interaction and to present new information to him. As such, *Front-End*, the graphical representation of CIDS, was analyzed. For best comprehension and structure, I have divided the explanation of this functionality on this component in smaller user-oriented functions, presented next.

Registration

Registration is the action that enables any user to become known to the system. It is a necessary step for anyone wanting to use most of the new functionalities that this work

provides. In order to record a new user into the system, several tasks were required to do first:

1. create an interface to gather user information;
2. create a work flow for the registration process to handle the new user's information;
3. create email template to send to the user emails;

This is the information about the user that I think is relevant for registering him on the system:

email It is used when CIDS needs to send emails to the user (for instance, when informing him of his new password);

password A random word composed of characters and numbers. In conjunction with email, enable the user to authenticate;

flag A boolean value indicating if user has completed the registration process or not;

token A randomly generated word. Technically, it is a 8-characters sized string, generated using a cryptographic hash function[1]. Every time this hash function is executed, it produces a fresh new random string, which I then store in a *token*. This token is used to create a challenge-response procedure between the system and the user, so that the former knows if the latter is performing a valid request or not;

From the list above, only the email is actually required by the user to input, the rest, besides the password, is just auxiliary information to help with the registration process. The interface created for this purpose can be seen on Figure 4.2. I also devised a workflow that "glues" this interface, previous list information and the sending email step:

1 User wants to register. He accesses the registration form interface (Figure 4.2), by following the "User Profile" hyperlink on home page (Figure 3.3) and then clicking on "Register";

2 He inserts his email;

3 The system records his email on the database, along with *flag* set to false and a randomly generated string for the *token*;

4 The system sends an email to the user (Figure 4.4), consisting of a URL containing, among other information, the recent generated token;

5 The user opens the email and clicks on the link;

6 The system generates a new *password* for the user and stores it in the database. The *token* is regenerated and *flag* is set to *true*;

7 Finally, the system sends its last email (Figure 4.5)to the user, containing user's credentials: *email* and *password*;

The newly created class, *User* (introduced during the analysis of the *database* component), is used to persist registration's data. Each datum has a matching table field: *email* is saved on *email* field, *password* on *password* field, *flag* on *is_registration_complete* field and *token* on *randomString* field.

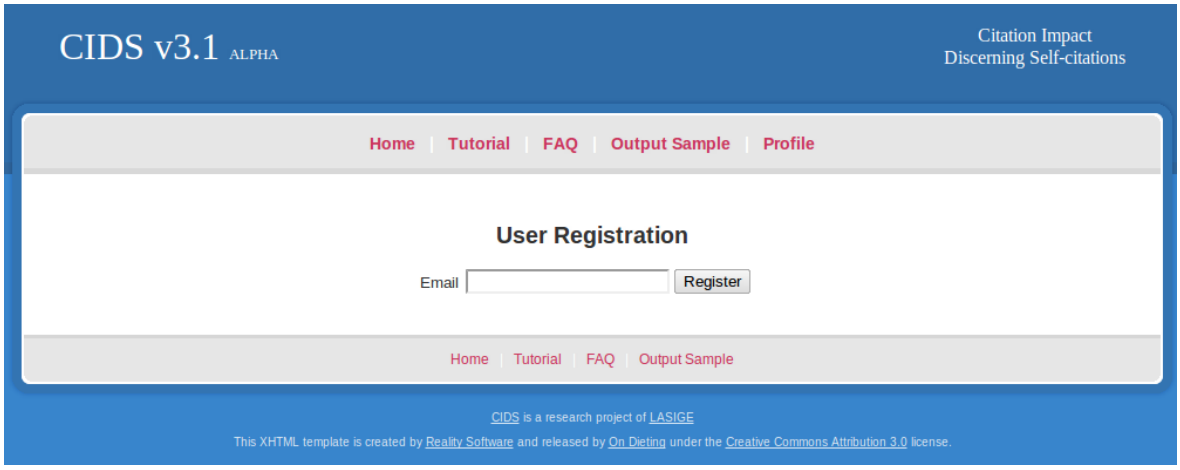


Figure 4.2: User profile registration form

This registration submission form (Figure 4.2) can fail in three cases: either when the user forgets to insert the email, when the email is already in use or, lastly, when email is incorrectly formatted. In all cases, a error message is always displayed to the user to inform him why submission failed. For instance, if the latter case occurred, the message illustrated in Fig. 4.3 would be shown.

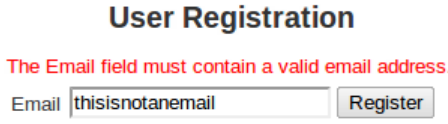
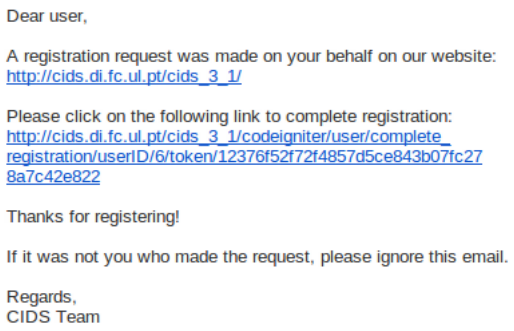


Figure 4.3: User registration error when providing a incorrectly formatted email address

The confirmation step is required on this process in order to prevent a user from blocking another one from registering. If this should happen, at worst the other user only receives an email that he was not expecting, which he can safely ignore.

From a security point of view, there is a step in the registration process that could be more secure and it is when the password is sent in clear-text to the user - step number seven. If someone is eavesdropping the communication channel between CIDS and user,


Dear user,

A registration request was made on your behalf on our website:
http://cids.di.fc.ul.pt/cids_3_1/

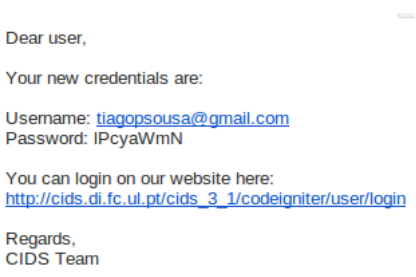
Please click on the following link to complete registration:
http://cids.di.fc.ul.pt/cids_3_1/codeigniter/user/complete_registration/userID/6/token/12376f52f72f4857d5ce843b07fc278a7c42e822

Thanks for registering!

If it was not you who made the request, please ignore this email.

Regards,
CIDS Team

Figure 4.4: First email sent to user during registration


Dear user,

Your new credentials are:

Username: tiagopsousa@gmail.com
Password: IPcyaWmN

You can login on our website here:
http://cids.di.fc.ul.pt/cids_3_1/codeigniter/user/login

Regards,
CIDS Team

Figure 4.5: Final email sent to user during registration and reset password

he (the eavesdropper) can actually read the password that is being sent. CIDS could use SSL certificates[2], where everything that is sent is encrypted first, which effectively disables this eavesdrop attack[3]. However, I feel that this higher level of security is not needed in a simple tool such as CIDS, because no user's sensitive information, apart from his email, is ever stored, so I did not employed it. For the record, because passwords are randomly generated, as opposed to being provided by the user, in the event of such attack no personal password is ever[4] stolen.

Login

Login is the action that enables a user to access his profile page, where he can see his queries. Given that *Registration* functionality already defined the database structure, what was necessary to do here was simply create an interface to gather user input and then perform some validation. The interface created can be seen on Figure 4.6, which is a typical login form in that there are two fields, username and password, albeit in this case the username is user's email. The credentials entered by the user are only considered valid when, first, they are found in same record in the database, and second, he has finished registration, that is, when flag *is_registration_complete* is set to *true* (on that record). If there is any form submission error, the user is presented with that error message, explaining why the form submission erred. In the case of login form, that only happens in the following three cases: missing email or password, bad email format (for instance, not containing the character '@') or the pair email/password is not found, that is, the credentials are invalid (Fig. 4.7). For security reasons, when the latter error occurs, it is not discriminated which of the fields - *username* or *password* - is wrong. If it was, a malicious user could potentially enter without permission inside a user's profile.

[2] A SSL (Secure Socket Layer) certificate secures the communication line between server and user, and also corroborates server's identity.

[3] As a side note, either sending the password via email or making the user defining it in CIDS would have the same eavesdropping problem, when SSL is not used.

[4] Except if the randomly generated password is the actual user's password.

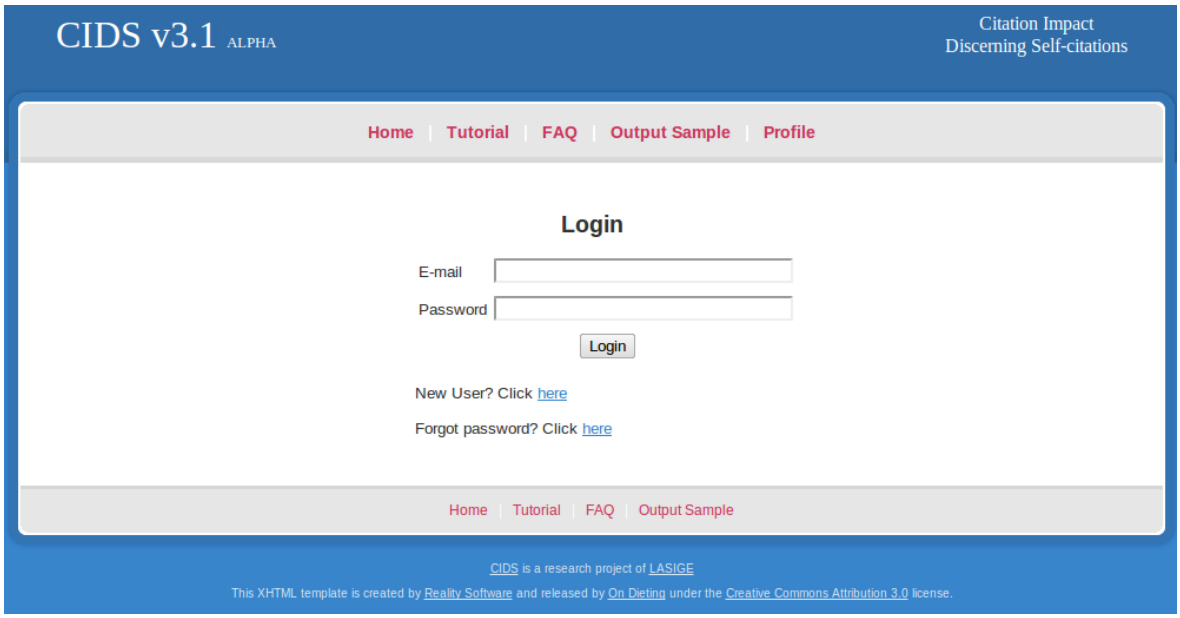


Figure 4.6: User profile login form

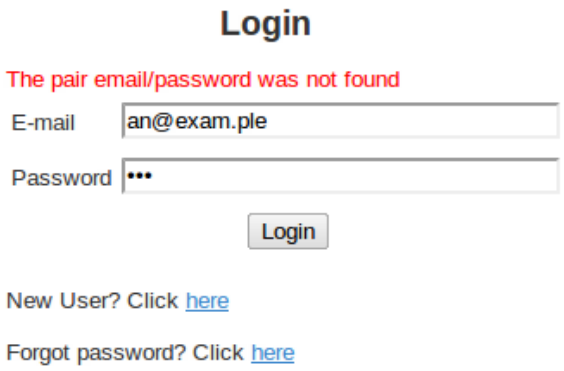


Figure 4.7: User login failing

Password Reset

Because the *Registration* process generates a random passwords, I believe that users will occasionally forget them, so I developed a mechanism that resets passwords and sends a new one to user's mailbox. Similar to *Registration* functionality, this one also required new user interfaces to collect input from the user, and a work-flow to handle for the entire *Password Reset* process. The interface created for collecting user input can be seen of Figure 4.8, which may recall *Registration*'s one in a way that only the title and button's label have changed. Next I present the work-flow created that handles the entire *Password Reset* process.

1. User wants to reset his password. He accesses the *Password Reset* interface by first clicking in "Profile" on CIDS' homepage (Figure 3.3) and then on "Forgot password? Click here" hyperlink;

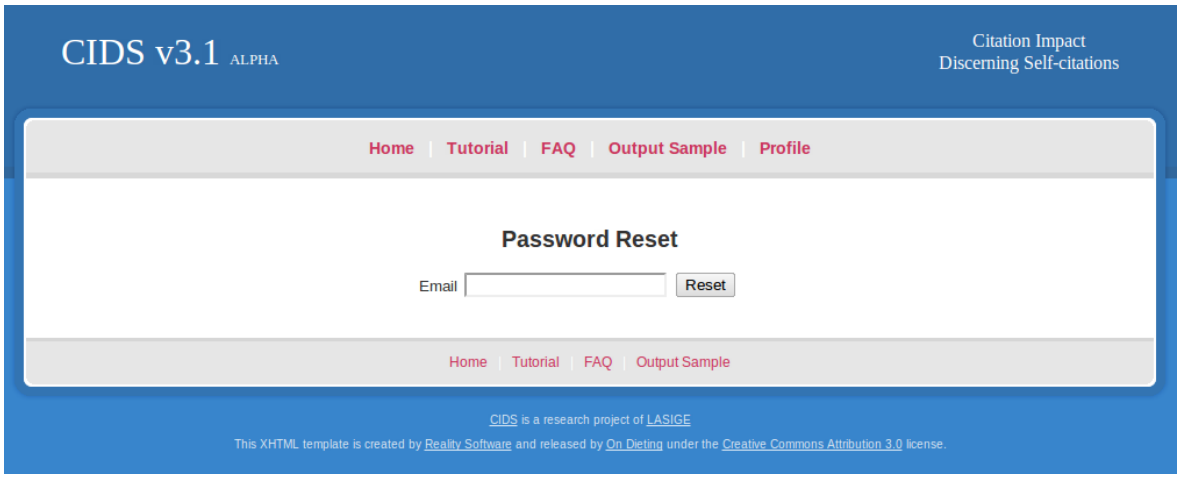


Figure 4.8: User profile password reset form

2. He inserts his email on the form field;

3. The system send an email to him, so that he can concur that he really wants to reset his password (Figure 4.9).

4. The user clicks on that link;

5. The system sends a newly generated password to user's email (and stores it in the database). It also regenerates the token *randomString* (introduced on *Registration* process) in order to be ready for use in the future;

6. User receives the password on his email and he can now login with his new credentials (Figure 4.10);

Dear user,

A request was made on your behalf on our website to reset your password.

Please click on the following link to receive a new password:
http://cids.di.fc.ul.pt/cids_3_1/codeigniter/user/complete_reset_pw/userID/1/token/5f043f25e67825498e0fbdccc258ea230cedd3f6

If it was not you who made the request, please ignore this email.

Regards,
CIDS Team

Figure 4.9: First email sent to user during password reset

Dear user,

Your new credentials are:

Username: tiagopsousa@gmail.com
Password: lPcyaWmN

You can login on our website here:
http://cids.di.fc.ul.pt/cids_3_1/codeigniter/user/login

Regards,
CIDS Team

Figure 4.10: Final email sent to user during registration and reset password

After the user inserts his email in corresponding field, CIDS checks if that email exists on the database, and if it does, sends an email to him. Also, similar to *Registration* process, unauthorized actions, which in this case would be resetting another user's password, are blocked due to the confirmation email (step number three) that is sent to the user.

Cockpit

Cockpit is the main deliverables of this *User Profile* functionality, it is where the user manages his queries. With that in mind, an interface was created to show all user's queries, which can be seen on Figure 5.4. The principal element on that interface is the table located in the center of the page, which contains all user's queries and has the following information about them:

Query The text of the query;

Subject The subject areas of the query that were/will be used to search for publications. These areas are chosen by the user when he creates the query;

Status This field can have different values, depending of status of the publication calculus. If publications calculus is taking place, it displays how many publications are left to complete the calculus, how many there in total and a percentage relative to those values. If it is not, the message "Publication filtering needed to start calculus" is shown;

Date completed The date of this query's last publication's calculus;

Actions If publication calculus has started, this field is empty. If the calculus is not taking place, then a button "Filter Publications" is displayed, allowing the user to filter query's publications. If publication calculus is either taking place or finished, an additional hyperlink is shown. Depending on publications calculus, this hyperlink navigates the user to either partial or final results;

On my preliminary report, I projected *cockpit* to have embedded a query insertion submission form, so that the user did not have to leave that interface if he wanted to create new queries. But I then started noticing that, if I were to create an embedded query insertion submission form where I wanted - on *Cockpit* interface - I needed to, first, find the code responsible for creating and handling that form and, second, duplicate and integrate it to where *Cockpit* was. However, after understanding how the code behaves (a task that required "untangling" the presentation and application logics) I found that it was easier to just simply insert the behavior I needed where the form was already than copy and integrate it on *Cockpit* interface. Additionally, if I had duplicated the code to another place, I would had to synchronize each change to that form on both locations, which, if forgotten, could lead to its own set of problems. Aside from having to remember that there were two locations to modify, it would be twice prone for introducing software *bugs*. This means that, contrary to what has been projected in my preliminary report, instead of a query submission form, there is a button labelled "Create New Query", which routes the user to the known query submission form present in CIDS's homepage.

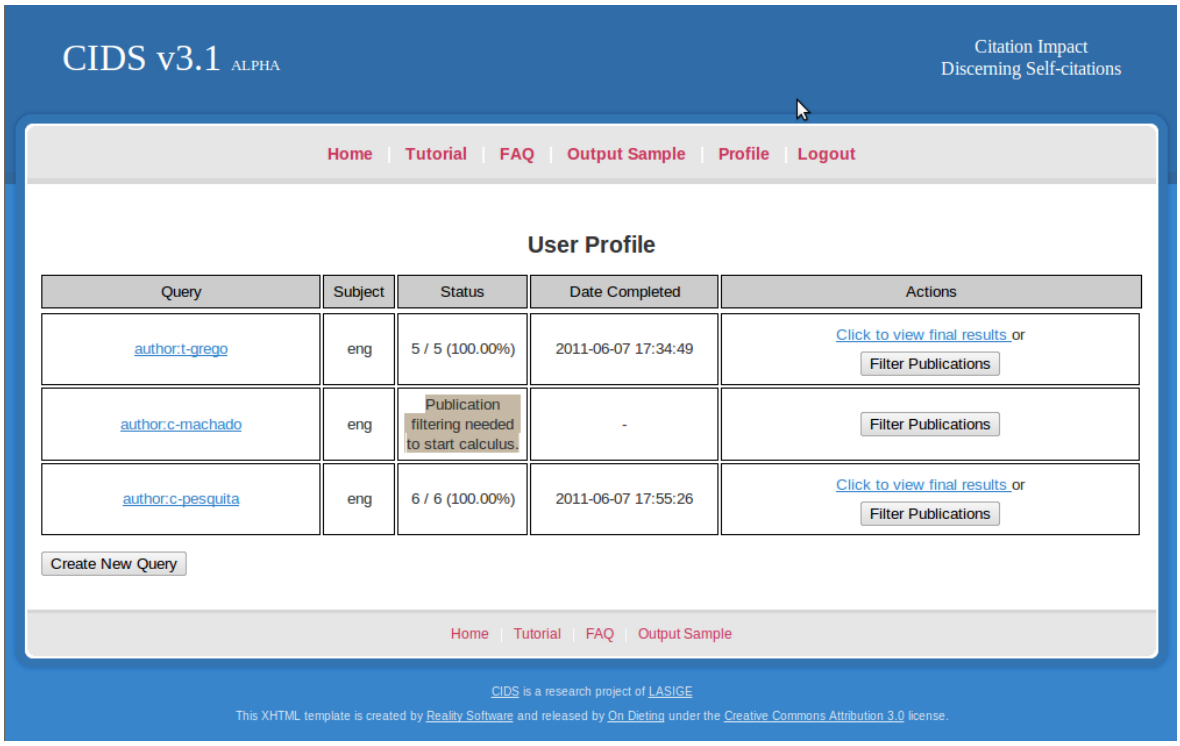


Figure 4.11: User profile cockpit, displaying user queries and their respective information.

As shown in *Cockpit* interface, the user is able to navigate to interfaces that already existed, namely, query creation, publication filtering and results.

The buttons present in Figure 5.4 drive the user directly to interfaces that already existed, namely, query creation, publication filtering and results. Some work had to be done so that the user could see those interfaces while retaining his session state, as well as interconnecting *Cockpit* to those interfaces.

In query creation, a normal user is required to insert his email. For a registered user, that should not be required, because he already has an email. In spite of not being needed, the email is still required to trigger subsequent CIDS flow, in this case, the email is needed in order to advanced for the next interface, which is publication filtering. To address this issue, I have generated a unique fictitious email[5] for each new query created by a registered user and then hid it (Fig.4.12), so that user does not misinterpret it and tries to modify it. The generation of this unique fictitious email is the key step in connecting *Cockpit* with query creation interface.

The next integration was with publication filtering. This interface was not designed so that it could be accessed via URL, only by access via query creation interface. And that is exactly what I did: I simulated a creation of a query and the click on a button, which then routes the user to the expected publication filtering interface. What happens "under the hood" in this simulation is that the query creation submission form is fed with

[5]Note that it is not this hidden email that will be used to communicate with user, but instead the one he provided upon registration.

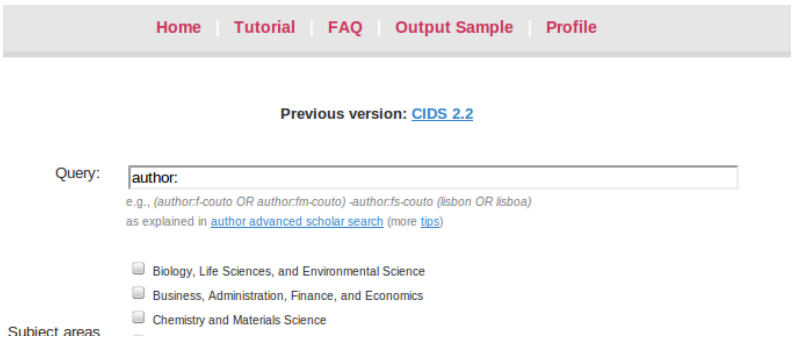


Figure 4.12: Email field is hidden in the query creation interface for a CIDS user

information about the query - text and subject areas. After that, one of the buttons present on that interface, "Calculate with Server" or "Calculate with Applet", is automatically "clicked". The "clicked" word here does not mean an actual mouse click but instead it executes the function that that click would trigger. I have chosen to "click" on "Calculate with Server" button by default, unless CIDS is blocked by Google Scholar; in that case, an asterisk, "*", is placed beside all queries entries on cockpit (Fig. 5.4) to symbolize that event and the "Calculate with Applet" button is used. After this, the publication filtering interface is finally displayed to the user.

The final integration was with results interface. Since there is no user input on this interface, the transition is done entirely by following an hyperlink. I have added an additional parameter to that hyperlink, called "profile", so that target destination - the results interface - knows that it is a registered user that is accessing, in which case hides the (random generated) email address on that interface.

During the integration with publication filtering interface, I performed a modification that I think that helps reduce disk space usage. CIDS always creates a new query for an author, even if the text of query already exists on his past queries: if an author (that is, an anonymous user) wants to re-filter publications of a query, CIDS creates one more query and copies all his preferences - the publications he has discarded in the past - to this new query. The problem is that, in addition to publication preferences, CIDS now stores citations preferences (more on this in Section 4.2) which exists in a much larger quantity than publications. As one might infer, replicating citations preferences of every publication for every query that the user decides to re-filter could easily fill the database with data that probably will never be used. So, instead of creating a new query every time user wants to re-filter publications, I have created a mechanism to detect if he is re-filtering a query or not. I did it by simply searching on his past queries, ones that have the same "text" as the query being created and if found, one of them is used instead. In a way, I am preventing query and publication (and citation!) to have history; they were never used anyway. Besides saving disk space, this modification simplified my work on the

next functionality, *Data Verification* (Section 4.2). Note that I only perform this detection for registered users; had I done it to regular ones, problems could arise since anyone can use another person' email and query, that is, anyone can erase other's preferences[6].

4.2 Data Verification

The objective of this functionality is to give the user the opportunity to verify and correct his bibliometric data. It is composed of two separate tasks:

1. Obtain and filter citations of publications, as well as change citation type;

2. Add new citations to a publication;

The ability to filter citations is one of the main objectives of this work. Similar to previous functionality, some CIDS components were required to be modified/extended. The *Database* component is necessary the first one that requires modifications, because it is where citations were going to be stored. Additionally, some rationale was needed in order to know how user's preferences of citations were going to be stored. This is discussed in Section 4.2.1.

Next is the *Queue of Requests* component. Citation is a new content type whose content is obtained querying Google Scholar. Since the *Queue of Requests* is responsible for accessing Google Scholar, this component needed to be modified because of this new content type. This component is discussed in Section 4.2.2. Related to *Queue of Requests* is the *Applet and Server Daemon* component, responsible for executing those requests, so this component was also analyzed (Section 4.2.4).

Both citation filtering and the addition of new citation tasks need applet support to be executed (for reasons that will be explained later). But when no applet support is found on client's browser, those tasks still need to be executed, otherwise some browsers could not benefit from this functionality. In order to accomplish this, the *Client-Browser* component was modified so that, even without applet support, it could still carry on those tasks. This is discussed in Section 4.2.6. To make possible the user to execute those tasks, new interfaces were created, meaning that *Front-End* component was modified, which is discussed in Section 4.2.5.

Transverse to all CIDS components is *Web-Services* one. Because a new content type of information - citation - is required to circulate among all components, new web-services were created. This is discussed in Section 4.2.3.

[6] Actually, that is what already happens: if someone uses another author's email and query, it can override that author's preferences. But because it is a new query that is created, the old values are stored in the database and could to be restored.

Next, each component of CIDS architecture is discussed, where, for each one, it is explained how it was modified to enable the execution the tasks mentioned in the beginning of this section.

4.2.1 Database

The first thing needed to be done was knowing what information about a citation should be stored. Here is the information that I think is relevant about a citation, both for its content and for integrating with other content type such as publication one:

title The citation's title;

link The citation's bibtex's link, used to retrieve the respective bibtex.

pub_id The identification (on the system) of the publication this citation cites;

type The type (self or non-self) of the citation;

year The year of the citation;

authors The authors of citation;

author_self The mutual authors between this citation and the publication it cites;

author_non_self The non-mutual authors between this citation and the publication it cites;

date_fetched The date this citation was fetched from Google Scholar;

Some of the fields above are for displaying purposes only, such as, the *title*, the *type*, the *year* and the *authors*. The *link* is a unique identification of the citation, which is also an hyperlink of that citation on Google Scholar. It is used for synchronizing the citations of publication with Google Scholar. The *pub_id* is used to know which publication this citations cites. The other ones - *author_self* and *author_non_self* ones - are for emphasizing on the interface which authors are self or non-self, to aid user. It is also important to store the following information about user preferences regarding citations:

discarded A true or false field, to know if a citation should be accounted in bibliometric result;

type The type of citation that user wants the citation to have. It can have the following values: self or non-self.

user_modifield A boolean field indicating if the user has modified the type of this citation. This is useful information to have. It allows an author to refresh the citations of a publication without losing his preferences;

I have chosen to store the information about citations in a class named *citation*. Next, I needed to choose how to connect this class to the existing database class diagram on Figure 3.2. I had two options:

1. I could connect *citation* straight to *query_pub*. The connection type could be one-to-many or many-to-many, depending if I wanted the citation bound to a specific publication of a query, or instead, sharing a citation among multiple publications. Remember that *query_pub* has a references for both *publication* and *query*;

2. Or I could individually connect *citation* to *publication* and then to *query*;

If I went with the first option then I would had to make an additional one, between using a one-to-many or many-to-many connection type. If I had opted with "one-to-many" connection type, an additional reference to *query* was required in *citation* class. That would effectively bind every citation to a specific publication of a query, that is, there could be several citations with the same title, link, year, and so on, replicated on the database, differentiating only in publication and query references. This seemed to me that it would waste disk space. If I had opted with "many-to-many" connection type, an additional linking class, between *citation* and *query_pub*, would have to be created. This one seemed a better choice than previous one, because information data was being duplicated. However, this approach can have a subtle problem. With this choice, I assume that, when storing preferences of a citation, there is already a record of a particular publication of a particular query on *query_pub* on the database. Well, but what if there is not one? I mean, what if this record is only created when user actually discards (or selects) the publication? In other words, what if the system assumes that a publication is not discarded by default, and acts based on that assumption? In this case, if the user starts interacting with citations before discarding/selecting publications, then the system would be unable to store user preferences. I have tried to find on source code code if such event ever happens, but because there were so many scripts to analyze, I decided not to take the risk and chose opted the second choice. The resulting class diagram can be seen on Figure 4.13. Observing previous image, connecting *citation* to *query_pub* perhaps makes more sense than connecting individually to *publication* and *query*. Unfortunately, for the reasons presented above, it would that is a risky choice. But the second choice does not come without its disadvantages too. One worst consequence of it is that it enlarged the SQL queries very much, which will be shown briefly.

As seen in *Front-End* (3.4), a user is able to change citations type and also discard them and as a result of that, his bibliometric results needed to reflect this change. Because these bibliometric results are a aggregation of several individual SQL queries, each one of those queries had to be modified to cope with citations value. For instance, the SQL query used for calculating the h-index is the following:

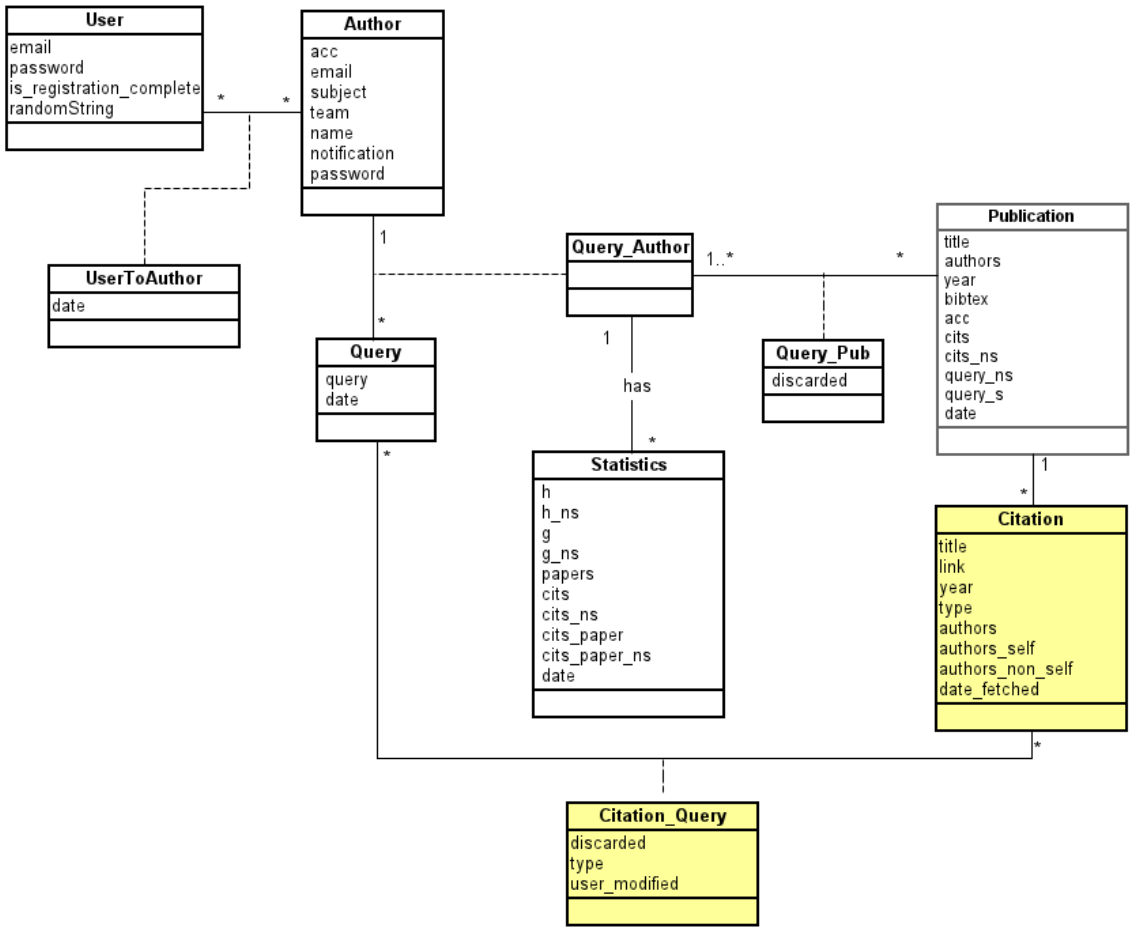


Figure 4.13: UML class diagram with classes related to Data Verification highlighted.

Code 4.1: Old SQL query for calculating the h-index of query of an author.

```
1  SELECT DISTINCT COUNT(p.p_id)
2  FROM query_pub AS q,publication AS p
3  WHERE q.a_id = $a_id  AND q.q_id = $_last_qid
4  AND p.p_id = q.p_id
5  AND q.discarded=
6  AND p.cits >= (SELECT DISTINCT COUNT(p2.p_id)
7      FROM query_pub AS q2,publication AS p2
8      WHERE q2.a_id = $a_id AND q2.q_id = $_last_qid
9      AND p2.p_id = q2.p_id
10     AND q2.discarded=
11     AND (p2.cits>p.cits
12     OR (p2.cits=p.cits AND p2.p_id>=p.p_id)));
```

But with the introduction of citations, it changed to:

Code 4.2: New SQL query for calculating the h-index of query of an author.

```
1  SELECT DISTINCT COUNT(p.p_id)
2              FROM query_pub AS q,publication AS p  JOIN
3              (select p.p_id, p.cits as cites
4              from publication p JOIN query_pub qp ON p.p_id=qp.p_id
                   AND qp.a_id=$a_id AND qp.q_id=$_last_qid
```

```
5             AND NOT EXISTS(SELECT * FROM citation c JOIN
                  citation_query cq ON cq.citation_id=c.id AND cq.
                  query_id=$_last_qid WHERE c.pub_id=p.p_id)
6         union
7         SELECT p.p_id, IF(citations.pub_id IS NULL,0,COUNT(citations
              .id)) as cits_ns
8             FROM publication p
9             JOIN query_pub qp ON p.p_id = qp.p_id
10            LEFT JOIN (SELECT c.id,c.pub_id,cq.*
11                            FROM citation_query cq
12                            JOIN  citation c ON c.id=cq.citation_id
                                  AND cq.discarded=    ) citations ON
                                  citations.pub_id =p.p_id AND
                                  citations.query_id=$_last_qid
13
14            WHERE p.p_id IN (SELECT DISTINCT c.pub_id  FROM citation
                   c JOIN citation_query cq ON c.id=cq.citation_id AND
                   cq.query_id=$_last_qid)
15        AND qp.a_id = $a_id
16        AND qp.q_id = $_last_qid
17        AND qp.discarded =
18        GROUP BY p.p_id) as all_cites ON p.p_id=all_cites.p_id
19    WHERE q.a_id = $a_id  AND q.q_id = $_last_qid
20    AND p.p_id = q.p_id
21    AND q.discarded=
22    AND all_cites.cites >= (SELECT DISTINCT COUNT(p2.p_id)
23                            FROM query_pub AS q2,publication AS p2 JOIN
24                                (select p.p_id, p.cits as cites
25                                    from publication p JOIN query_pub qp
                                        ON p.p_id=qp.p_id  AND qp.a_id=
                                       $a_id AND qp.q_id=$_last_qid
26                                   AND NOT EXISTS(SELECT * FROM
                                       citation c JOIN citation_query
                                       cq ON cq.citation_id=c.id AND cq
                                       .query_id=$_last_qid WHERE c.
                                       pub_id=p.p_id)
27                                union
28                                SELECT p.p_id, IF(citations.pub_id IS
                                      NULL,0,COUNT(citations.id)) as cits
29                                   FROM publication p
30                                   JOIN query_pub qp ON p.p_id = qp.
                                      p_id
31                                   LEFT JOIN (SELECT c.id,c.pub_id,cq.*
32                                                  FROM citation_query
                                                     cq
33                                                  JOIN  citation c ON
                                                     c.id=cq.
                                                     citation_id AND
                                                     cq.discarded=
                                                      ) citations ON
                                                     citations.pub_id
                                                      =p.p_id AND
                                                     citations.
                                                     query_id=
                                                     $_last_qid
34
```

```
35                              WHERE p.p_id IN (SELECT DISTINCT c.
                                    pub_id  FROM citation c JOIN
                                    citation_query cq ON c.id=cq.
                                    citation_id AND cq.query_id=
                                    $_last_qid)
36                              AND qp.a_id = $a_id
37                              AND qp.q_id = $_last_qid
38                              AND qp.discarded =
39                              GROUP BY p.p_id) as all_cites2
40                          ON p2.p_id=all_cites2.p_id
41                  WHERE q2.a_id = $a_id
42                  AND q2.q_id = $_last_qid
43                  AND p2.p_id = q2.p_id
44                  AND q2.discarded=
45                  AND (all_cites2.cites>all_cites.cites
46                  OR (all_cites2.cites=all_cites.cites AND p2.
                        p_id>=p.p_id)));
```

This SQL query has three properties:

1. It takes into account only publications that are not discarded;

2. It takes into account only citations that are not discarded;

3. If user has not filtered citations then this query output is equal to the old one, but if he has filtered citations, previous properties taken into account on the bibliometric output;

The last property is the one responsible for enlarging the SQL query in Code 4.1; the reason is explained next.

If we observe Figure 4.13, we can see that the class *publication* has the field *cits*, which is the number of citations of that publication. This field was used in previous h-index SQL query (Code 4.1) for calculating this bibliometric. But now, citation count can be determined by using the *citation* and *citation_query* classes, so they should be used instead. This is true except in one case. When the user has not starting filtering citations, these new classes should not be used, because they do not contain any citation data at that moment. The SQL query in Code 4.2 is aware of this situation, which is mainly reason why it is so big. Similar to h-index, all SQL queries that calculate the remaining bibliometrics (h-index discerning self-citations, g-index, g-index discerning self-citations and total citations) were also modified. Those queries can be seen in Appendix C.

The default type chosen, between "self" and "nonself", for citations was "self". The explanation is on Section 4.2.4.

4.2.2 Queue of Requests

Like as been mentioned in the beginning of this section, the *Queue of Requests* has the responsibility to fetch information from Google Scholar. Current architecture has request types for each content type it fetches from Google Scholar (for example, publication) and because *citation* is also a content type, this component needed to be modified. After thinking about how to extract the information I wanted of Google Scholar, I decided to create these new request types:

cite_total Fetches the raw information of all citations of a publication. It is analogous to the *html_raw* request type for publications;

cite_bibtex Fetches the bibtex of a citation. It is analogous to the *bibtex* request type for publications;

When *Queue of Requests* was created, CIDS only handled publications content type, which made me wondering if it could handle these new requests too. After beginning to understand how it operated, I realized that the citation concept is nearly the same as publication's (the difference being that a citations cites something, a publication does not), meaning that I could integrate these new request types in *Queue of Requests* easily. So now the queue of requests has the following request types: *html_raw*, *bibtex*, *cite_ns*, *cite_total* and *cite_bibtex*. Section 4.2.4 explains how these new requests relate to each other and to new ones that will be introduced on that section too, as well as how citations are processed throughout CIDS components.

4.2.3 Web-Services

Nearly all CIDS components use web-services to communicate with each other. Since there is a new content type, it make sense to create web-services so that it can be reached by all components. To that end, new web-services were created. Most of them were created as they were needed during the development of component *Applet and Server Daemon*, discussed in Section 4.2.4. As such, they are presented first so that the reader gets an idea of has been created and then they are used during the explanation of the component *Applet and Server Daemon*.

I have divided the description of the web-services over five categories: citation, publication, settings, stack and user, which can be seen on Appendix A. Next is shown an example of a singe web-service description (WS means Web-service).

WS Name setType;

WS Method POST;

WS Description Defines the type of a citation. The type is modified only if user has not done it himself (except if force_op is true). This web-service should be invoked after fetching new citations from Google Scholar;

WS Parameters

id The identification of the citation. Not optional;

type The type (self/non-self) wanted for the citation. Not optional;

force_op If this parameter is true, then citation type is always modified. Optional;

queryID The identification of the query of publication that citation belongs to. In conjunction with force_op, is used to modified the citation type already changed by the user. Optional;

WS Return Value None;

4.2.4 Applet and Server

With database ready to store data, queue of requests ready to handle citation content type and web-services ready to exchange citation information, and was lacking was work flows that integrated all previous components in performing the tasks mentioned in the beginning of this *Data Verification* functionality. The name of those algorithms/work flows are: "fetch citations" and "add extra citation".

As we saw in 3.5, both *Applet* and *Server* are the executors/handlers of the requests of the *Queue of Requests* and perform similar actions. Since both are implemented in Java programming language, that allowed me to encapsulate those algorithms on reusable classes for each executor. Those algorithms are explained next.

Fetch Citations

The main idea behind this algorithm was to fetch, store and display citation information. Here is a representation of the algorithm (which is almost "copy-paste" of the code I have created), written in pseudo-code, followed by a explanation:

```
    FetchCitationTask(  int_t pub_acc,
                        int_t author_acc,
                        int_t query_id,
                        output_t output) {
1.          startFetching();
2.          fetchCitesLinks();
3.          fetchAndProcessCitesBibtex();
4.          fetchPublicationBibtex();
```

```
5.          changeCitationsType();
6.          terminateFetching();
7.          updateCitationsTypeOnOutput();
    }
```

This algorithm receives the following information on its method signature:

pub_acc The identification of a publication;

author_acc The identification of the author who started this process;

query_id The identification of the query the publication belongs to;

output The place where citation information is displayed. When is applet that it is executing, the output is the web page.

The first line, `startFetching()` function calls the web-service *startFetching*, which, as the name suggests, starts the process of fetching citations. It "grabs" a request from the queue of requests with type *cite_total* belonging to publication *pub_acc* (if it is not found, a new one is created) and sets it's state to pending, to be available for later processing. It also sets the request *bibtex*, of the same publication, to *pending* state because CIDS needs to know the authors of the publication in advance, to later access citation type, and that request supplies that kind of information.

In the second line, `fetchCitesLinks()`, the *cite_total* request above mentioned is processed: the HTML retrieved from this request is parsed, resulting in the creation of requests with type *cite_bibtex*, one for each citation found. Those requests have their state set to *pending*, for immediate processing.

In the third line, `fetchAndProcessCitesBibtex()`, the requests with type *cite_bibtex* mentioned above mentioned are processed. Basically, the corresponding bibtex of each citation is fetched from Google Scholar and stored in the database. If is the applet that it is handling those requests, after finishing executing each one of them, it displays citation data on the interface.

In the fourth line, `fetchPublicationBibtex()`, the *bibtex* request type that was set to *pending* on line 1 is executed: the bibtex of the corresponding publication is fetched from Google Scholar and stored in the database.

In the fifth line, `changeCitationsType()`, the *cite_ns* request type, automatically created after the *bibtex* one is processed; it queries Google Scholar for citations with type "non-self" and also queries the database for existing publication's citations. If it finds matching citations between these two sets, the second one have their type changed to non-self (remember that the default type of a citation is "self"). It also sets this publication's status from *processing* to *ready*, meaning that all publication resources have been processed and they are ready to be accounted for in bibliometrics result.

In the sixth line, `terminateFetching()`, the whole process of fetching citations is terminated. The web-service `terminateFetching()` is used to do that. Internally, what it does is setting all citations discard state to "no".

Finally, in the seventh line, `updateCitationsTypeOnOutput()`, the type citations are refreshed on the screen. This only happens if it is the applet that is executing; the server daemon does not output.

When using applet, the user is notified of the progress of the fetching citation process, which is discriminated in four stages: *queued* when another fetch citations task is taking place, *pending* when no fetch citation tasks are in line for execution, *processing* when effectively the citations are being fetched from Google Scholar and stored internally, and, finally, *complete* when citations are displayed to the interface.

Add extra citation

Similar to fetch citations, albeit much simpler, I developed an algorithm for adding an extra citation to an existing set of citations of a publication:

```
    AddExtraCitation(  int_t pub_acc,
                       int_t author_acc,
                       int_t query_id,
                       string_t bibtex_url,
                       output_t output) {
1.        startAddingExtraCitation();
2.        fetchAndProcessCitesBibtex();
3.        terminateAdding();
4.        appendCitationOnOutput();
    }
```

This algorithm receives the same parameters as fetch citation task's ones, plus one, the *bibtex_url*. As we will see in *Front_end* component of this functionality, the user is required to input a bibtex URL in order to add a citation to one of his publications. This bibtex URL is what is used in previous algorithm. For example, the following bibtex URL:

`http://scholar.google.pt/scholar.bib?q=info:rmrOOaCvnOIJ:scholar.google.com/&output=citation&hl=pt-PT&as_sdt=0,5&ct=citation&cd=0`

can be used for add a new citation to a publication. The user obtains this type of URL by performing a Google Scholar search with "Show links to import citations into Bibtex" property turned on, on Google Scholar's preferences interface[7].

[7] http://scholar.google.com/scholar_preferences?hl=en&lr=&output=search

In the first line, `startAddingExtraCitation()`, the process of adding the extra citation is started; A request with type *cite_bibtex* is set to *pending* state and has the URL provided by the user.

In second line, `fetchAndProcessCitesBibtex()`, that request is processed and as a result, the newly fetched citation is associated with current publication.

In third line, `terminateAdding()`, this task is terminated. It sets the default values for citation type and discard status, respectively, to *self* and *no*.

Finally, in fourth line, `appendCitationOnOutput()`, if the applet is executing, the citation content is appended to citation list on the interface.

Similar to fetch citation task, the user is notified of the progress of this task on the interface.

Applet detection and communication

Both applet and server daemon execute the algorithms mentioned before. By interacting with the interface, the user triggers those tasks.

When there is applet support on the browser, it makes sense using it, because server daemon is shared across all users and the applet is not. After some research, I have found a tool[8] that is capable of detecting applet support on browsers. It is also capable of invoking methods of the applet from Javascript, although it has some restrictions (mentioned later).

But what should be done when no applet support is detected on the browser? When that happens, CIDS must resort to user's browser itself to carry on with the tasks. This is was done with AJAX calls (more on this in Sec. 4.2.6, when discussing about the "Client Browser" component).

When applet support is detected on the browser, the typical interaction between applet and interface is the following: The user presses a button which triggers a specific applet's method, which, in turn, communicates with the server using web-services. During, and at the end of, that interaction, the applet returns values to browser, which, in turn, displays those to the user.

The restrictions I previously mentioned are related to browser security. When Javascript invokes a method within a applet, the permissions associated with that request are minimum, for instance, it can not can access the network (that is, the Internet). After some research, I found a way to overcome this issue, which was instantiating an ordinary object inside applet's scope, prior to Javascript calls and then delegate those calls to that object. Because this object was created by the applet itself during its inception, if Javascript invokes applet's methods through that object, it has applet's restrictions and if we sign the applet, then there are no restrictions.

[8] http://tech.chitgoks.com/2010/05/07/detect-if-jvm-exists-using-javascript/

4.2.5 Front-End

For a user to be able to filter citation and add new ones, he needs something to interact with. This section describes what interfaces were created so that the user could perform the tasks/actions mentioned in Section 4.2.4: fetch (and filter) new citations and add new ones.

Figure 4.14: Old publication filtering revamped: it has a new button labeled "Fetch Citation".

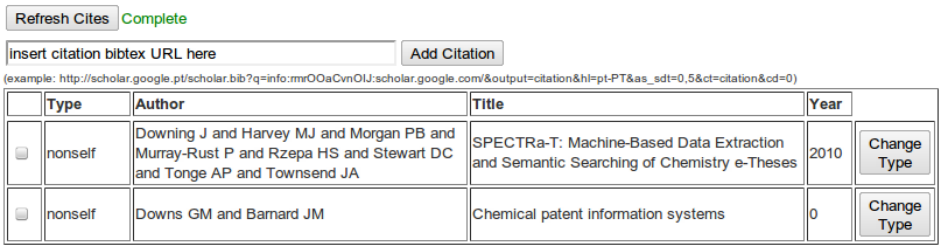


Figure 4.15: An example of list of citations that appears after pressing the "Fetch Citation" button

When comparing Figure 4.14 with filtering publication interface of Fig.3.4 it can be spotted a new button on each publication record, labeled "Filter Citations". It is this button that triggers the "Fetch Citation" task (Section 4.2.4). After pressing this button, a list of citations eventually appears (Figure 4.15). For each citation, user can either discard it (check-box on the left of citation's type) or change it's type (the button on the right side). If the latter happens, the citation's type label changes accordingly. If year is not found in citation's bibtex, then a 0 (zero) is shown.

Next to button labelled "Fetch Citation" is a label that shows the progress of the operation that is executing, whether if it is *fetch citation* or *add extra citation* task that it is executing. This label changes sequentially, starting on "queued", then "pending", then "processing" and finally, "complete" (whose meaning is mentioned in Section 4.2.4). For instance, in Fig. 4.15, we can see that the *fetch citation* tasks has completed.

After fetching citations, user can trigger the *Add Extra Citation* task. He does that by first inserting a valid citation's bibtex URL in the respective input form, which is the one with the placeholder text "Insert citation bibtex URL here" and then clicking on the "Add Citation" labeled button (Fig. 4.15). If the URL provided is invalid, a warning box the shows up indicating that error (Fig. 4.16).

While either task is executing, the table where citations are displayed, as well as all input and buttons, are temporarily blocked, to prevent unwanted actions.

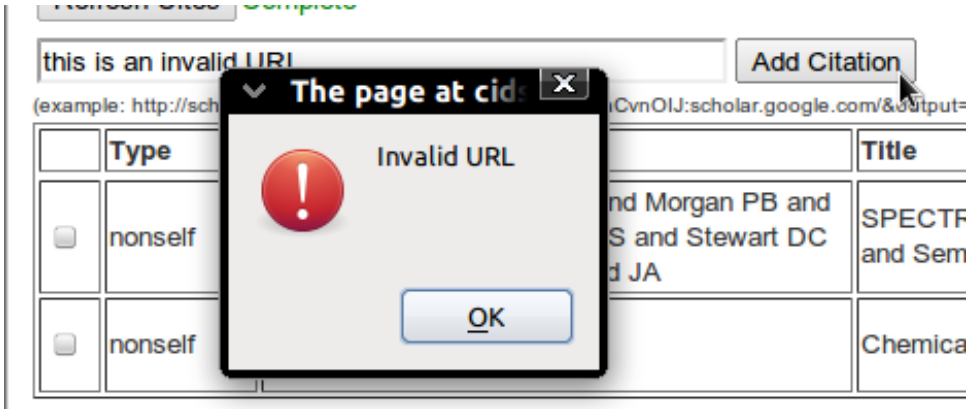


Figure 4.16: A warning box appears when an invalid URL is inserted while adding an extra citation.

When user accesses this interface for the first, the browser usually asks him if he allows applet execution, which he must respond affirmatively in order to use it. This is the applet discussed in previous section and this is the default behavior of most browsers.

The typical HTML button causes a page refresh. If the page refreshed after pressing the button on this interface, then all interface modifications during user interaction (for instance, the table that lists citations) would disappear[9]. I have changed this button behavior by attaching customs functions handlers. Those functions are in fact AJAX calls to CIDS web-services, that triggers the algorithms discussed in previous section. So, instead of a page refresh, the user sees content appearing on the the interface instead.

4.2.6 Client Browser

Like it was said before, when no applet support is found on the browser, the later still needs to carry on the execution of the tasks mentioned in 4.2. To be able to do that, I have recreated, using Javascript[10], the algorithms of those tasks. Since most of the steps in those algorithms use web-services - which are accessible by all CIDS components, including this one - the recreation was quite straight-forward. For the "Fetch Citation" task, the recreation was simply calling the web-service *startFetching*, and periodically checking if that task had ended using the web-service *hasFetchingEnd* (a complete list of web-services can be found on Appendix A). Analogous for the "Add Extra Citation" task, it was a matter of calling the *startAddingExtraCitation* and *hasAddingExtraCitationEnded* web-services, respectively. Since this approach uses *Server Daemon* (which, like has been said, handles requests of all users), these tasks can take a very, very long time to complete.

[9] Although that information remained stored in the database.
[10] Javascript is the browser's "programming language"

4.3 Group Analysis Updated

If we recall from Section 3, to perform a group analysis task, one needs to perform a couple of sequential steps, being forced to wait between those steps. Also, one does not get notified of the results when they are ready to be viewed, that is, when the tasks finishes.

As the name of this section states, the objective here is to improve what already exists. That objective is decomposed on the following requirements:

1. The user should be able to start a group analysis task and leave his computer while the task continues to execute, that is, no further presence of him should be required to complete the task;
2. The user should be notified when the group analysis completes;
3. The user should also be able to bookmark the group analysis's web-page for future reference;

The first requirement can be solved by creating a program that does what the user would do, but in an automated manner. Besides figuring out how to do this, I also had to decide in what programming language should this program be made, because, as mentioned before, CIDS was made with two server-side programming languages:, Java and PHP. The second requirement is also solved with this program, because if we know when to exit that program, we know when to notify the user that his group analysis task has finished. The third requirement is solved by simply displaying to the user an URL, generated according to the input file he provides and which he can then bookmark it.

Since the first requirement involves creating a program, the component of CIDS that must change is the *Applet and Server Daemon* one. The reason why it is this component is simple: this program needs to continue executing in the background and Java is better than PHP at it. Also, since this program is server-sided (executes on the server, instead of on client) and it should done in Java, I have putted it on that component (4.3.2). The second requirement is part of the first one, as such, it was inserted on *Server and Applet* component too. I added an additional text information, which is a percentage of group analysis completed calculus completed, meaning that I had to modify the user interface. The third requirement is a simply presenting an URL on the user interface, which also means modifying the user interface. Both modifications are explained on the next section (4.3.1).

4.3.1 Front-End

This functionality required to notify the user when his group analysis task completed, as such, the first modification I made on the existing interface of group analysis from (Figure 3.11) was creating a new input field for the user to provide his email to receive

the notification (Figure 4.17). If the email provided is valid, then the file he uploaded is parsed just like it was in the past. After uploading the file, the control of the flow of the application is passed to PHP, which, in turn, launches the JAVA program (introduced in the the beginning of this section and discussed in Section 4.3.2) to execute in background, and returns the control back to the user along with an URL, so that he can bookmark it (Figure 4.18). The third requirement is thus reached.

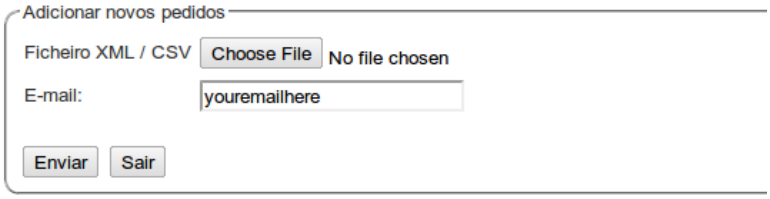


Figure 4.17: New field - email - in file upload form, for notification purpose.

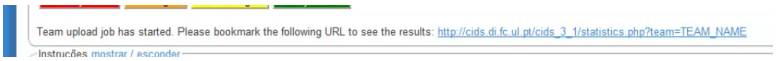


Figure 4.18: An example of the link that is displayed after submitting the CVS file. The text "TEAM_NAME" is a placeholder for the team name.

Like it is mentioned in the beginning of this section, I have improved the team results interface by introducing a numeric percentage of group calculus progress (Figure 4.19). It is a simple count of all pending requests belonging to all members of the group (whose name is provided in the file uploaded) divided by the total number of requests of those members. This indicator disappears once it reaches one-hundred percent, which symbolizes the end of the calculus. After this, the notification is sent to user (this is explained more detailed in the next section).

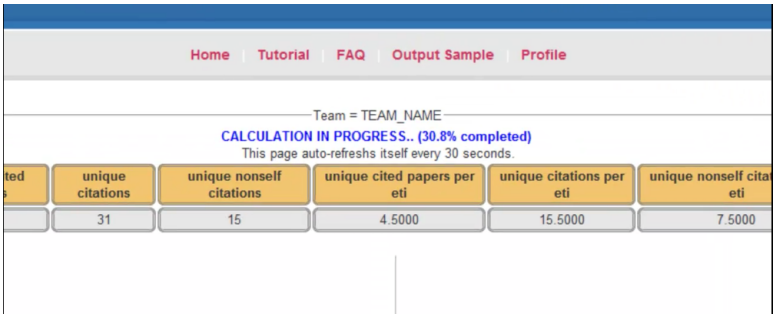


Figure 4.19: The current percentage and the indication that this team calculus is still in progress.

4.3.2 Applet and Server Daemon

The three steps that the user has to perform in order to initiate a group analysis task, after uploading the file containing the members of the group, is:

1. Wait for requests with type *html_raw* (the request responsible for fetching raw publication information) to reach *ready* state, that is, wait until all members' publications are fetched;

2. Use the administration interface to find those requests (now with *ready* state) and press a button on the interface so that bibtex requests, of the publications mentioned above, are created;

3. Find, again using the administration interface, all those bibtex requests and set them to *pending* state, to be processed;

There is not any user decision on this sequence, which means that the above steps can be automated.

After the user entered his email and the processing of the CSV file took place, I had access to all authors' identification information belonging to the group as well as group's name and user email. Having authors' identification number and group's designation, I could find and filter all the requests of the group and with email I could notify the user when results were ready. The next imperative action was to pass all this data from PHP to Java, and second, to keep the later executing in the background. I decided to pass, first, the group name, then the email, and finally a list of authors' identification number, for easy argument handling on Java side. Next, inside the Java program, I had to simulate user interaction step.

Before explaining how I have done this simulation, I need to clarify one aspect about *Server Daemon* component. This component is actually composed of two parts: the main one, that processes requests of the *Queue of Requests* - the *Server Daemon* - and another one, "invisible", also used by the later, that executes specific routines. These are simple functions but must be done in Java because of the heavy work (in terms of memory usage) they do, for example, parsing huge HTML documents. In this case, after the user submits the CSV file and email as well, the server launches a Java routine that will continue executing in background until it finishes.

To simulating the first step, I have put Java querying the database if all *scholar_raw* requests of the group had their state equals to "ready" periodically, until they did have. I used the `get_daemon_stack` web-service to retrieve those requests. After that I check if prior group analysis job is executing. This can happen, if, for instance, a user mistakenly submits the same file twice, either due to a browser's glitch or using the "back" button present on the browser interface [11].

If a prior group job is not running, then step two can start. This one finds all *scholar_raw* requests, belonging to the group, with state *ready* and invokes a routine responsible for creating *bibtex* requests for each publication of each *scholar_raw* request found. Since

[11] I do not check the case where two users decide to input two files with same team name; because this functionality is not open to public, is not likely to occur

there was not a *web-service* that returned multiple *scholar_raw* requests at once, I have "created"[12] one.

The third step was simply putting these new *bibtex* requests into *pending* state, for later processing[13].

With all the above steps automated, it is now possible to perform a group analysis task without any user interacting from the moment he initiates one. The system still needs to wait for all group requests (both *bibtex* and *cite_ns*) to reach the "ready" state; once that happens, a notification - an email with the team's URL - is sent to user's email and the group upload task terminates.

Surprisingly, one simple, yet vital, step had to be done, without which no group statistics would appear on group analysis result interface. I found out that the individual's statistics of each author, from which group statistics are derived, were all empty. Later, I found out that they were only saved to the system when someone actually visited each member statistics' URL. An odd behavior but with a simple solution; because the URL of the statistics page of a member can be accessed programmatically, this last step was simulated by simply performing a request to all members' URL. After doing this, the team results finally appeared on the group results interface.

After these three steps and this last one that fixes authors' statistics, the notification is finally sent to the email address that was provided by the user who had started this group analysis task. For historical and debugging reasons, each task/job is logged to a file.

4.4 Auxiliary functionalities

Although the title refers "functionalities", these were actually more of a guideline/ directives that help building previous functionalities. I have taken into consideration two important aspects of a web application during the development of previous functionalities: security and cross-browser compatibility. They had to be taken into consideration during the development and not after because they are not just separate modules that can be integrated but instead, they actually modify how previous functionalities are implemented. The reason why these functionalities are explained on the same section is due their relatively small content when compared with previous ones.

4.4.1 Security

One of the security measures used was the *token* used on *User Registration* and *Password Reset* functionalities. That token generated generated with random characters, it is

[12] Actually I did not create one *per se*, but instead converted a n existing script to act like one; the end result is roughly that same, though.

[13] Similar to previous step, I converted a script to act like a web-service.

refreshed every time it is used and every user has one. When sending URLs to a user with this token, the system knows that only that user can respond to it. This is done to prevent a malicious user to impersonate a regular CIDS user, except in one case, which is when the malicious user steals that URL or the password that is sent. This situation can happen if he is sniffing (that is, eavesdropping) the network communication channels. If a malicious user eavesdrops and succeeds at stealing user email and password, he could impersonate that user. Usually, SSL technology is used to prevent this type of attack. SSL encrypts communication between client and server, preventing a malicious user from eavesdropping the "conversation" between them. But using such technology implies overhead of additional verifications and content encryption in communication channel, inherent of SSL, slowing down communication speed. However, since no sensitive information is stored, such as as an address or credit card number, having his URL stolen does not really harm him. Moreover, the password never is a personal one. This means that CIDS would not greatly benefit if used SSL technology.

I have mentioned one type of attack that is done against web applications. But there are much more, which CIDS is susceptible to, for instance, SQL injection. This type of attack is ranked number one in the attack category list of web applications (where CIDS is included). It consists in injecting malicious code inside the SQL queries that query a database. The objectives of this type of attack can range from reading private information to wiping out information of repositories. It was also one of the reasons that motivated me in using a framework that I have used in the past, called *CodeIgniter*[14]. It is a PHP framework (much like jQuery is for Javascript), that has already a set of functions that prevent SQL Injection (among other types of attack). Unlike previous CIDS code[15], written in raw PHP, I have simply used the framework's built-in functions which renders this attack useless, by parameterizing queries. For instance, let's imagine CIDS has the following SQL query to obtain information about a user:

```
SELECT * FROM user WHERE id = '.  $user_id;
```

and the variable `$user_id` does not have any input validation before it is used. If a malicious user is able to inject the following value into that variable:

```
'; DROP user
```

it becomes:

```
SELECT * FROM user WHERE id = ''; DROP user
```

which may delete all users from the database. What *CodeIgniter* does to prevent this is by having a white-list of characters. If it detects a dangerous character on the value of a variable used on a query, like `'` (apostrophe), it escapes it to `\'`. This way, the character cannot modify query's original intention. At worst, it can create a query that does not returns results, but in that case, the applications must be prepared to respond to that event.

[14] http://codeigniter.com/

[15] which, by the way, is vulnerable to this type of attack

With the introduction of a user restricted area, CIDS needs to records session when users login. With the use of session comes a common web application vulnerability, called *Session Fixation*. On this type of attack, the malicious user tries to define himself the identification number of a normal user, in order to impersonate him. But since *CodeIgniter* abstracts the developer (that is, myself) from using the directly the *session* variable and regenerates session identification in shorts periods of time, it protects the application from this type of attack.

CIDS has a particular advantage over the common website with database; almost all its content can be obtained from Google Scholar, meaning that it is very difficult to lose its content. Only user preferences can not be restored if, for instance, CIDS database was deleted.

4.4.2 Cross-Browser

Cross-browser is when a website works as expected on all major browsers (such as *Chrome*, *Firefox*, *Opera* and *Internet Explorer)*. An important step in achieve full cross-browser compatibility is avoiding Javascript functions that only work on a few browsers. Because there are lots of functions available, knowing if each one is cross-browser compatible is very difficult. To alleviate this task, several frameworks were created, such as *Prototype* and *jQuery*, claiming cross-browser operability.

When I started developing for CIDS, *Prototype* was the framework being used. Previous CIDS developer warned me that *Prototype* was not cross-browser compatible, though claiming to be, since some of its functions did not work in some major web-browsers, such as *Internet Explorer*. I took his word for granted and decided to use another framework, *jQuery*, that I have already used it in the past and worked flawlessly. So, *jQuery* was used on the development of new CIDS functionalities.

In the beginning of this report, I mentioned that new CIDS features would be cross-compatible. However, a few design decisions made along the way, have invalidated this claim. For instance, to enable communication between Javascript and applet, required for giving fast feedback response to the user, I have partially sacrificed cross-browser compatibility. This happened because some browsers did not support this kind of communication. To amend this, I created a mechanism that detects if applet support is available on a browser and if it is, then Javascript communicates with applet and vice-versa (and the applet communicates with server), but if not, then it communicates with server using web-services, through AJAX calls.

Even the communication between Applet and Javascript is different because it has its set own problems, for instance, in detecting applet support in browser and detecting if it can communicate with Javascript too. I believe that with more time, I could enhance this mechanism by detecting more efficiently this kind of situation and automatically falling back to using web-services if such support is not found.

As I have said, CIDS was built using *Prototype*. After some research, I decided not to migrate existing code from *Prototype* to *jQuery*, because such task would require a great amount of time to complete because *Prototype* is used in many places and also because it has functions whose purpose I did not found in jQuery. This means that existing, non cross-browser compatible, CIDS features, remained that way. These problems (Javascript-applet communication and full migration from *Prototype* to *jQuery*) are aggravated when the same browser behaves differently in different operating systems (such as *Windows*, *Linux* and *Mac OS*). I believe that, If the three previous cross-browser problems - Javascript-Applet communication, *Prototype* to *jQuery* migration and different behavior of the same browsers in different operating systems - were resolved, CIDS could be full cross-browser compatible. We can see in Table 4.2 that users using *Internet Explorer* are nearly a fifth of total users who accessed CIDS in August 2011. Since *Prototype* does not work well on this browser, those users are unable to use CIDS properly.

Browser	Visits	% visits
Firefox	235	51.20%
Chrome	102	22.22%
Internet Explorer	80	17.43%
Safari	35	7.63%
Opera	4	0.87%

Table 4.2: Accesses to CIDS by browser, in August 2011

4.5 Miscellaneous Work

This section lists the work I did that does not fit on previous functionalities description. Besides doing my work, I also have done others things during my work, such as, maintain a previous CIDS version (3.0) (Section 4.5.1), develop a module to enhance CIDS operability (Section 4.5.2) and extend the old CIDS API to a new one using a different technology (Section 4.5.3). The remainder of this section discusses these three topics.

4.5.1 Maintenance of CIDS 3.0

Before and during the development of previous functionalities, I had to maintain CIDS 3.0. Some of that work of what I did is listed next:

- Migrated CIDS 3.0 to a new machine which I configured from scratch - both installation and configuration of the operating system. This machine was also going to house CIDS 3.1 eventually;

- Created a new video tutorial to reflect changes between CIDS 2.2 (the version prior to 3.0) and 3.0;

- Helped creating scripts for starting, stopping and restarting the *Server Daemon* Java program and also a *crontab*[16] to restart that program, periodically;
- Helped fix an important SQL query about authors and their publications, which were being calculated incorrectly;
- Created indexes on CIDS 3.0 database to improve the speed of some SQL Queries which were taking more than on minute to execute.
- I tried to enhance existing source code that was written Java, by employing OOP[17], as opposed to the procedural paradigm that currently existed;
- Fixed a *Javascript* table sorting problem;

There was particular problem that happened during the first months of my work, when I only had little knowledge of how CIDS operated. From one moment to another, CIDS was not being able to fetch bibtex of publications from Google Scholar. After a couple of days investigating, I come to conclude that Google Scholar had modified a critical aspect that CIDS relied. In order to bibtex links appear in Google Scholar search results, CIDS must explicitly request them. Prior to this event, what was done was setting Google Scholar's preferences via HTTP request, which in turn returned a cookie that then was used in the next requests to Google Scholar. It was that cookie that enabled bibtex links to appear on Google Scholar search results, which CIDS uses. If they do not appear, then CIDS stops working altogether. After a few more days of testing and searching, I found someone who had a similar problem[18] and his solution was appending a specific string, ":CF4", to a specific *cookie*, "GSP", that Google Scholar received. This little "hack" allowed bibtex links to show up again in search results.

4.5.2 Pro-activity

CIDS uses Google Scholar to fetch and store publication information, and (now) citations. It needs to perform multiple requests to Google Scholar in order to obtain that kind of information. However, Google Scholar has a self-defense mechanism that blocks its access when detects rapid automated access to its publication repository. During the development of my work, I noticed that CIDS was getting constantly blocked, more than usual, by Google Scholar, perhaps due to some modification in their detection mechanism. When this situation happened, instead of stop performing requests, CIDS simply continued doing it, and, worst, it was blindly storing whatever Google Scholar returned as a result. For instance, when Google Scholar blocks, CIDS requests return with "HTTP

[16] http://livecronjobs.com/
[17] Object Oriented Programming.
[18] http://blog.venthur.de/2010/01/27/query-google-scholar-using-python/

Error 403 Forbidden" message and that was what was being stored. Besides stealing the focus, this was corrupting the database with bad data and when that happened, I needed to quickly fix this problem, so what I did was: kill the Java program executing in background of the server and put the requests that failed to *pending* state. Because I was developing the new CIDS functionalities while performing those tasks, I needed a permanent fix for this.

What I did was developing a mechanism that made CIDS aware of Google Scholar blocking event. It consists of periodically checking if Google Scholar is blocking requests, by performing a dummy request and then analyzing its response content; if it finds the message "HTTP Error 403 Forbidden", or similar, it aborts CIDS, that is, kills the Java program. After that, when Google Scholar (eventually) stops blocking, which can be known by periodically analyzing the dummy request content, it re-revives CIDS by re-launching the Java program in background. The information for knowing that Google Scholar is blocking CIDS is stored on the database. This is necessary because of the existence of two request executors that access Google Scholar - the applet and the server - that need to know if they can handle a request at a specific time. To know that, both executors query the database (using a web-services which I created for that purpose) if Google Scholar is blocking or not.

The creation of this mechanism prevented me from having to constantly doing the tasks above mentioned in the event of Google Scholar blocking event.

Albeit it took effort to develop, it spared me from having to constantly fixing the database with saved me from constantly having to fix the database data.

4.5.3 Complementary API

CIDS 3.0 web-services, which I address by "API", were written in PHP, without any frameworks' support. After seeing the source code of this API, I noticed that it was somewhat difficult to comprehend because there was no distinction between database (SQL queries), business (how requirements were implement) and presentation (displaying XML) logic domains. On top of this, the new features I had to develop required the creation of even more web-services. This forced me in creating another API, but instead using just PHP, I used a framework (mentioned in Section 4.4.1). This framework plus a module[19], facilitates immensely the creation of web-services, queries to database and XML handling for output. The only downside of this approach is that I ended up with two API, each one with it's own URL, instead of just one, which separates web-services set in two. But since only the new features use the new API, and both API are for internal use, there is not really a problem with having two separate API.

[19]http://net.tutsplus.com/tutorials/php/working-with-restful-services-in-codeigniter-2/

Chapter 5

Results

The main result of this work is a product; it is a new version of CIDS, designated "3.1", composed of functionalities that I have developed and also a installation guide for installing and configuring CIDS.

This chapter's structure is similar to previous one, where each each sections discusses a specific functionality. The difference now is that each section, instead of being divided by each CIDS component, is composed of what that functionality provides to the user, by describing and showing what it can do, as well as the impact it had in CIDS structure and on future development. For instance, for *Data Verification* functionality, even tough it affects all CIDS components (as seen on Table 4.1), what is demonstrated here is the fact that the user is able to fetch and add citations, plus the impact the functionality had during and after its implementation.

Due to its size, the installation guide of CIDS 3.1 is shown in Appendix D. To complement this chapter, several mini-videos demonstrating how to use the new version of CIDS are included on attached CD.

5.1 Data Verification

The objective of this functionality is to allow the user to have more control of his bibliometrics results by allowing him to filter the citations of his publications. The result of this functionality is graphically reflected on an interface well known to the regular CIDS user: the publication filtering interface (Figure 5.1).

Comparing the old with the new one, the latter [1] has a new button labelled "Filter Citations" on each publication row (Figure 5.2). Once that button is clicked, a list of citations eventually appears, similar to what is shown on Figure 5.3. For each citation, it is possible to discard it (far left check box) and change its type (far right button). The text in the "type" column, as well as the button's label, indicate the citation's type and both

[1] Which now should be called "publication and citation filtering interface", but I will stick with the first designation.

Publication Number	Title	Reference (Authors)	Citations
01	MULTIPLE DISC BRAKE WITH RESILIENT RELEASE MEANS	TE Grego - US Patent 3,486,588, 1969 - Google Patents	06
02	Identifying Gene Ontology Areas for Automated Enrichment	C Pesquita, **T Grego**... - Distributed Computing, Artificial ..., 2009 - Springer	03
03	Identification of chemical entities in patent documents	**T Grego**, P Pezik, F Couto... - ... , Soft Computing, and ..., 2009 - Springer	02
04	A portable dielectric constant sensor based on time of flight measurements	**T Grego**, M Dionigi, A Moschitta... - ... Conference, 2009. I2MTC' ... - ieeexplore.ieee.org	01
05	Identifying bioentity recognition errors of rule-based text-mining systems	FM Couto, **T Grego**, HP Bastos... - ... , 2008. ICDIM 2008. ..., 2008 - ieeexplore.ieee.org	01

Figure 5.1: Old publication filtering table

change when that button is clicked. The input box to add new citations to this publication is located above this citation filtering table. That input box accepts links similar to a Google Scholar bibtex hyperlink (an example is shown beneath the input box). After filtering and discarding publications and/or citations and initiated the calculus, the result can be seen at the usual place, which is the results page.

Publication Number	Title	Reference (Authors)	Citations
01	MULTIPLE DISC BRAKE WITH RESILIENT RELEASE MEANS	TE Grego - US Patent 3,486,588, 1969 - Google Patents	06
Filter Citations			
02	Identifying Gene Ontology Areas for Automated Enrichment	C Pesquita, **T Grego**... - Distributed Computing, Artificial ..., 2009 - Springer	03
Filter Citations			
03	Identification of chemical entities in patent documents	**T Grego**, P Pezik, F Couto... - ... , Soft Computing, and ..., 2009 - Springer	02
Filter Citations			
04	A portable dielectric constant sensor based on time of flight measurements	**T Grego**, M Dionigi, A Moschitta... - ... Conference, 2009. I2MTC' ... - ieeexplore.ieee.org	01
Filter Citations			
05	Identifying bioentity recognition errors of rule-based text-mining systems	FM Couto, **T Grego**, HP Bastos... - ... , 2008. ICDIM 2008. ..., 2008 - ieeexplore.ieee.org	01
Filter Citations			

Figure 5.2: New publication filtering table

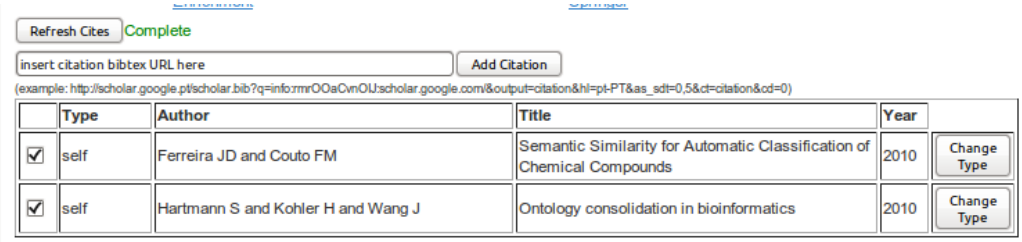


Figure 5.3: A zoom in the list of citations
A zoom in the list of citations of a publication

Unintentionally, this functionality also improved CIDS internally (from a developer's viewpoint):

- By having to create more requests type to handle citations (for instance *citation_bibtex*), it made me develop new classes for the old and new requests type. While creating

those classes, I noticed a similar behavior in them, so I created a base class and had the others (the request types) extended it. Now, if a developer what to create a new request type, he simply needs to extend that base class and then add the behavior he wants, without having to worry about the rest. In programming terms, this is base class is called "inheritance" and is consider a result because it structured the source code immensely;

- Having a a new content type - citation - to handle, introduced by this functionality, I needed to create more web-services for CIDS components be able to use it. As you may recall from Section 4.5.3, I was not very happy with how the API was implemented at that time, so I created a new one using the framework *CodeIgniter* which greatly helps with the coding part. If it was not for this functionality, I would never have created this new API and since is much more structured (due to the framework) than previous one, it is easier to add new web-services, so I also consider it also a result of my work;

5.2 User Profile

As stated in the beginning of this work, the objective of this functionality is to allow the user to manage several queries simultaneously and track their progress. The principal graphical component of this functionality is the "cockpit", discussed in Section 4.1.2 and can be seen on Figure 5.4.

Analyzing previous picture, we can see that for each query, we can know the query that was/is/will be used on Google Scholar to retrieve publications, the subject areas, the progress of the calculus of each query (if it has started), possible actions we can do with that query - filter publications or view results - and at the bottom a button to create new queries. If we follow the "Filter Publications" button, we are taken to the new publication filtering interface, shown in previous section (Figure 5.2), if we follow the results page, then we are taken to a regular results page, if we follow the "Create New Query" button, we are taken to the create query interface with the modified submission form (Figure 5.5).

Before accessing this "cockpit" page, the user must first register. He does that on registration interface which is accessed by going to CIDS homepage, then follow the "Profile" hyperlink, then "New User? Click here" link. When on the page, he must provide his email and have it confirmed it over email. This work flow as already been discussed earlier in Section 4.1.2 and exists a similar one for password reset, but is used when the user wants to retrieve his forgotten password. When the user success and either registering or resetting his password, he logs in, using the login interface (Figure 5.6). After that he is taken to the "cockpit" page, where he can perform what already has been mentioned before. Contrary to previous functionality, this one did not lead me into developing additional modules.

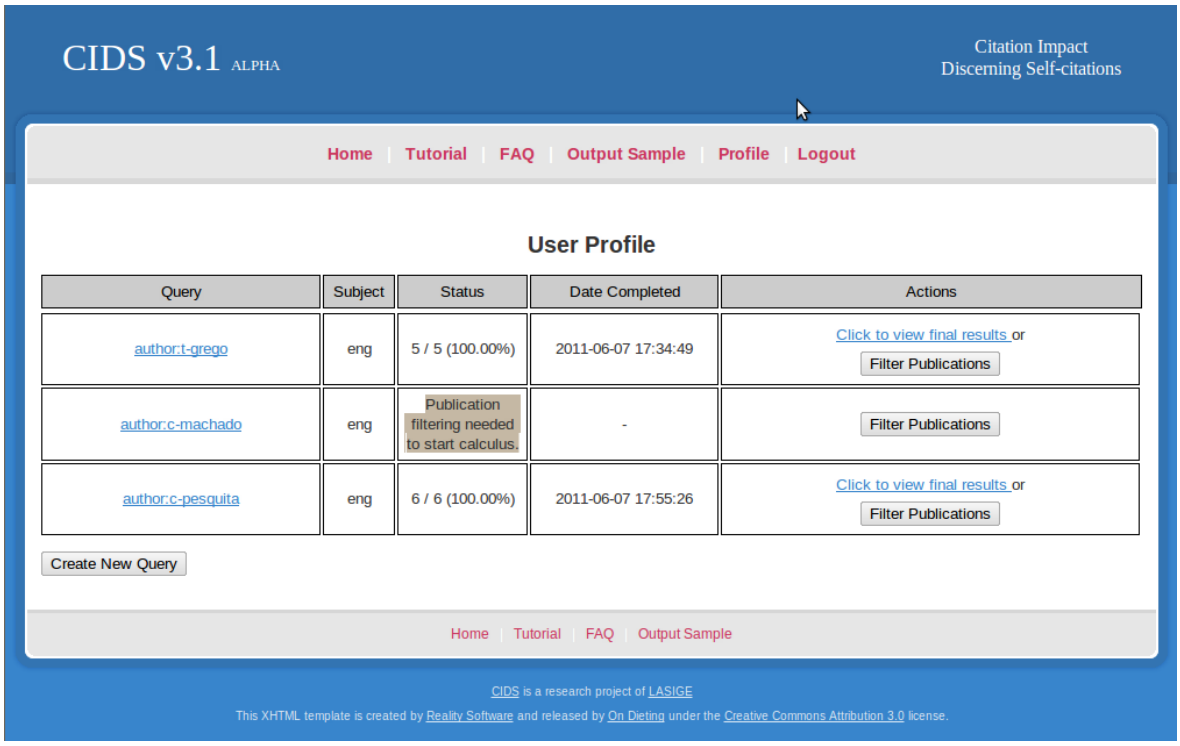


Figure 5.4: User profile cockpit, displaying user queries and their respective data.

5.3 Group Analysis Updated

As mentioned in Section 4.3, the objective of this functionality is to automate the process of calculating bibliometrics of a group of authors. To that end, a slightly modified interface from the original one (Figure 3.11) was created, which can be seen on Figure 5.7. This interface has text box so that the user inputs his email to receive notifications at end of the calculus. After he submits the file, an URL is presented, beneath the colored buttons, for him to bookmark it, which is a direct link to the bibliometrics results page of his team. This latter page was also slightly modified (as mentioned in Section 4.3.1) with the inclusion of a progress indicator, so that the user knows how much of calculus has been completed so far. This can be seen in Figure 5.8. Each job/task is an executing Java program, meaning that it can be aborted by the administrator if he wishes to. Also, each job progress (which step the job is at and what it is doing at the moment) is logged to a separate file.

This functionality also did not lead me into developing additional modules.

5.4 Miscellaneous Work

In my opinion, the most notable "miscellaneous" result of this work is the pro-activity behavior CIDS now has (discussed in Section 4.5.2). Basically, instead of "waiting" for Google Scholar to block, CIDS now takes an active role and goes "ask" Google Scholar

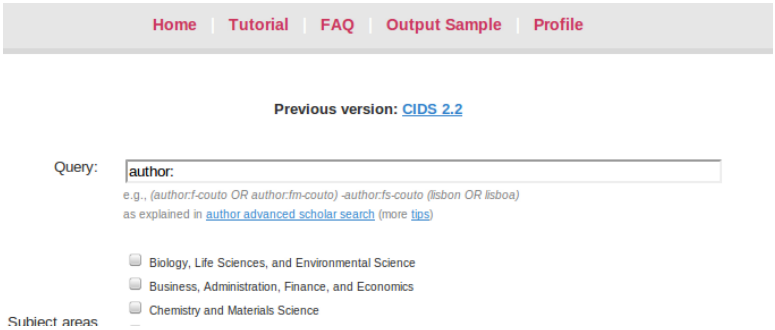


Figure 5.5: Modified CIDS homepage for creating queries
The email field is not display as it contains a random generated email address.

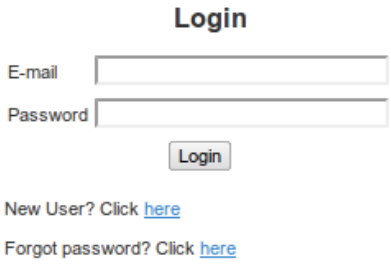


Figure 5.6: Login form.

if it is blocking[2]. If it is, then it does two actions: (the server) stops making more requests and notifies all other components that the access to Google Scholar is temporarily blocked. The first action is done by simply killing the running Java program, and the second is changing a record on the database (I made applet check this record prior to starting making requests to Google Scholar). Additionally, a warning text is displayed on CIDS homepage (Figure 5.9).

The best part of this is that, if Google Scholar stops blocking, CIDS automatically revives the Java program and changes the record on the database, effectively not needing any intervention from the administrator. Currently, CIDS "asks" Google Scholar if it is (or has stopped) blocking from thirty to thirty minutes. This was done by simply creating a cronjob which executes a specific PHP script created specifically for this. The periodicity can be change on that cronjob.

All CIDS code was put under in source control software called *Google Code*[3] (Figure 5.10). This means that CIDS source code is viewable by everyone and everyone can suggest new functionalities or fixes for errors found. The link is http://code.google.com/p/cids/.

[2]Actually, it is just makes a dummy request, but the intention is really to know if Google Scholar is blocking or not.

[3]*Google Code* is the source control versioning system created by *Google*. More information on https://code.google.com/

Figure 5.7: The new team upload interface

The new team upload interface, with an extra text box and URL that appears once the user submits his file

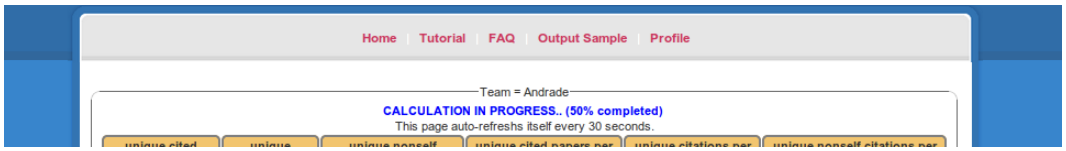


Figure 5.8: The group's calculus progress.

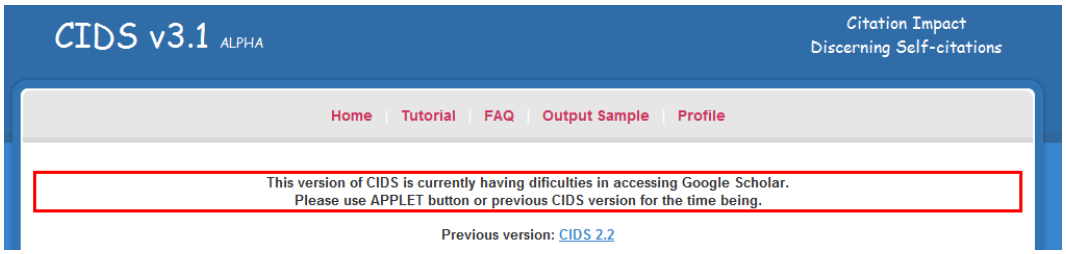


Figure 5.9: CIDS displaying a warning sign when Google Scholar blocks

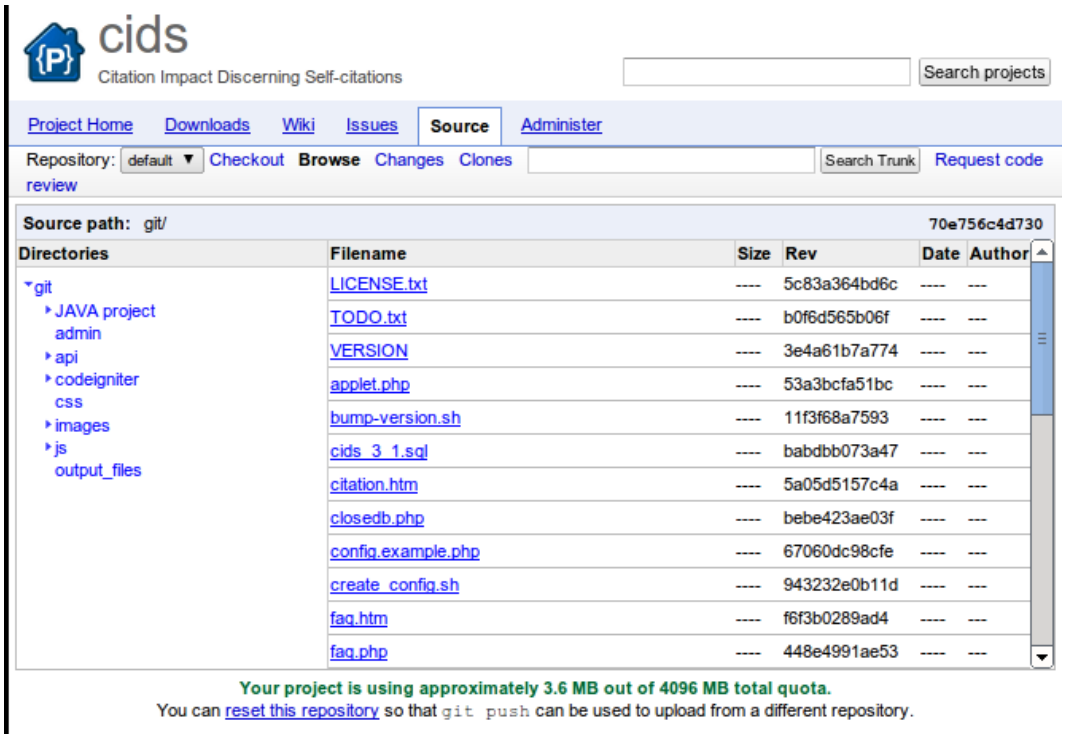


Figure 5.10: CIDS source code on *Google Code*.

Chapter 6

Conclusion

Looking back at the objectives in Section 1.2, I can say that, even though I have not completed them all, I feel good about what I did with CIDS. This new CIDS is better than previous one because:

- It enables filtering citations and also change their type. This helps improving the quality of the data retrieved from Google Scholar. In spite of being a helpful feature, it should be noted that citations type can also be changed with the wrong intentions, for instance, to tamper bibliometrics results (for example, to increase the h-index). As such, when using results originated from CIDS to compare authors, I would advise to corroborate those results with a manual evaluation;
- It allows users to have multiple queries calculated simultaneously. This is useful, for example, to someone who needs to calculate several author's bibliometrics. In previous versions, that kind of calculus had to be done sequentially, which implied that the user, first, knew that each author's bibliometrics calculus had finished to start the next one, and second, had to wait between those successive calculus, which could take a very long time to complete (depending on server load and/or browser capabilities to support applet technology). Another way of performing such calculus was using several random emails, one for each bibliometrics result, but in that case the user would not be notified of some calculus (unless he possessed several emails). In current version, he only needs one email and, after registering on CIDS, which takes a minute long, it is capable of calculating several queries at once;
- It is capable of detecting when Google Scholar is blocking it. When it detects that situation, it automatically aborts the program responsible for fetching data from Google Scholar, thus avoiding the storage of bad formatted data. Once it detects that it can fetch from Google Scholar, it revives the program;
- Although invisible to the end user, the source code developed of CIDS is now a little more structured (more classes with well defined responsibilities), which facilitates future development, debugging and testing;

I do not know if it is a coincidence but Google Scholar has recently launched a tool called *Google Scholar Citations* and describes it as "a simple way [...] to compute your citations metrics and track over time"[1]. It cannot be known for sure if tools such as CIDS were part of the reason why Google Scholar launched this tool in first place, though there are a few some similarities on user interface between their tool[2] and CIDS, for instance, the publication table (Figure 3.7), the graphic bar over the years (Figure 3.6) and the table with bibliometrics (Figure 3.5). Even if *Google Scholar Citations* did not get its inspiration from CIDS, it is very satisfying seeing a giant in computer industry developing to an area of interest where CIDS was pioneer.

Google Scholar's tool is currently in a limited launch to a small number of users. I think that, if such tool ever sees daylight, CIDS will not be able to compete with it, for the following reasons:

- *Google Scholar Citations* does not have to wait to have access to it's own publication and citation database;
- It also has directly access to data, whereas CIDS needs to parse lots of HTML documents to obtain the same data;

Only time will tell if such tool overthrows the rest.

As one might notice, there was a planned functionality that was not done, the "Data Validation". Like it was mentioned in objectives (Section 1.2), this functionality purpose was to automatically filter publications and citations for the user. It would use an external tool to detect patterns in user filtering preferences and automatically perform predictions of which publications and citations to discard. The reason why it was not done is presented next.

During my first semester, besides doing my thesis, I was also taking two course units. Because those course units had projects of their own, I was not able to work full time on my thesis and, as such, it was inevitable that some functionality would have to be left behind. By the time I noticed that there was not sufficient time to do the rest of functionalities I had left to do, "User Profile" and "Data Validation", I had to choose between these two. I was more comfortable doing the first than the second one because I already had done a similar to project on a previous course unit; the second functionality was something completely new for me and I was unable to estimate how much time it would consume. I ended up choosing the first functionality due to time constraints, although I must confess that "Data Validation" functionality sounded more interesting.

As for future work, I personally would like to see for CIDS:

[1] http://googlescholar.blogspot.com/2011/07/google-scholar-citations.html
[2] http://scholar.google.com/citations?user=kStFxtkAAAAJ&hl=en

- The main buttons, "Applet" and "Server", merged, making the "applet" concept completely hidden from the user. I partially developed this when I created the mechanism that detects if a browser can support *applet*. I think that with more time and investigation, this merge would be implemented successfully;

- Separate presentation from business logic. I tried (and succeeded in some cases) doing this, but it a very task consuming time. Because CIDS uses *Java*, there are some scripts that have PHP mixed with HTML and also *Javascript*, making very hard to debug the code, so separating those logics would greatly enhance development;

I also personally believe that, once the new tool of Google Scholar, *Google Scholar Citations*, reaches the public, because it has the *Google* brand, people will immediately start using it and ignore existing tools. In spite of this, I think that CIDS is a very capable tool for doing what it proposes to and it should be used.

Appendix A

List of Web-services

This appendix lists the web-services created during the development of functionalities "User Profile" and "Data Verification".

WS Name duplicateCiteToPub

WS Method POST

WS Description Given a citation bibtex link and a publication identification, duplicates citation content to that publication citation's.

WS Parameters

pubACC The identification of Google Scholar of the publication. Not optional.

bibURL The URL (base 64 encoded) of the bibtex of the citation to be duplicated. Not optional.

WS Return Value XML with the status of this operation, along with citation info.

—

WS Name find

WS Method GET

WS Description Returns all citations belonging to a given publication and query identification.

WS Parameters

pubACC The identification of Google Scholar of the publication. Not optional.

queryID The identification of a query. Not optional.

WS Return Value XML with a list of citations.

—

WS Name toggleDiscard

WS Method POST

WS Description Toggles the discard state of a citation.

WS Parameters

citationID The identification of the citation.

queryID The identification of the query.

WS Return Value XML stating if the operation was successful or not.

—

WS Name startFetching

WS Method POST

WS Description Start the fetching citations task for a given publication.

WS Parameters

pubACC The identification of Google Scholar of the publication.

authorACC The identification of the author starting this task.

WS Return Value XML stating if the task was started or not.

—

WS Name hasFetchingEnd

WS Method GET

WS Description Checks if the fetching citation process has ended for a given publication.

WS Parameters

pubACC The identification of Google Scholar of the publication.

authorACC The identification of the author who wants to know about this task.

queryID The identification of the query of the publication that has the citations.

WS Return Value XML with a boolean value depending if the process has ended and also all citations of that publication.

—

WS Name terminateFetching

WS Method POST

WS Description This web-service marks the end of the fetch citation process. It associates all fetched citations to a given query. It should be called after web-service hasFetchingEnd response status equals TRUE.

WS Parameters

pubACC The identification of Google Scholar of the publication.

authorACC The identification of the author who wants to end this task.

queryID The identification of the query of the publication that has the citations.

WS Return Value XML with a boolean value stating if the task has been terminated successfully.

—

WS Name hasAddingEnd

WS Method GET

WS Description Checks if a the bibtex of a citation of a publication (started by an author) has been fetched. If it has, returns the corresponding citation.

WS Parameters

pubACC The identification of Google Scholar of the publication.

authorACC The identification of the author who wants to end this task.

bibURL The URL (base64 encoded) of the citation that is being added.

WS Return Value XML stating the status of this task and also citation information.

—

WS Name terminateAdding

WS Method POST

WS Description Similar o terminateFetching, but only associates a single citation to a query.

WS Parameters

citationID The identification of the citation.

queryID The identification of the query.

WS Return Value XML stating the status of this task.

—

WS Name startAddingExtraCitation

WS Method POST

WS Description This web-service starts the task of adding a citation.

WS Parameters

pubACC The identification of Google Scholar of the publication.

authorACC The identification of the author who wants to start this task.

bibURL The URL (base64 encoded) of the citation that will be added.

WS Return Value XML stating if this task has started or not.

Appendix B

Map of dependencies of scripts

This appendix was used to aid me in detecting which functions assume that an author has only one query. It was created during the development of "User Profile" functionality.

Most of the nodes in the map has the following syntax: <*file_name*>::<*function_name*>, to aiding me in locating better function in source code. The icons display throughout the map have the following meaning:

- The green check mark was used to assess which functions were already analyzed;
- The stop sign means that the analysis on that branch stopped (no more references to others functions);
- The warning icon means that either the respective function was already reviewed or it references another that was already reviewed (a loop);

The filename also has it's extension explicitly written because it helped me knowing which programming language was being used. Finally, due to the large width of the map, I have partitioned it in five smaller images, displayed next.

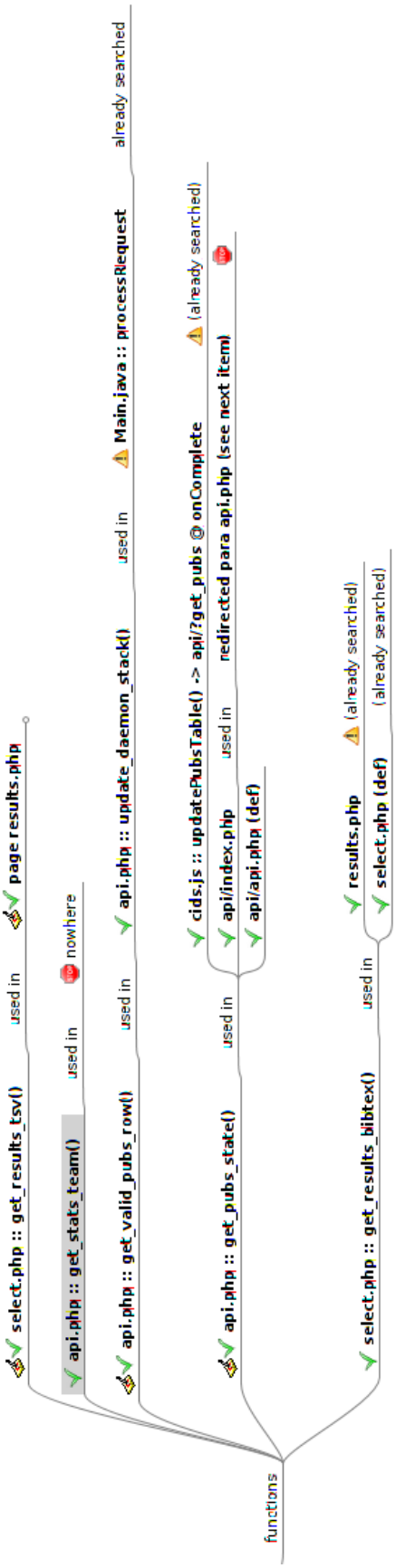


Figure B.1: First image (of five) of the map of functions.

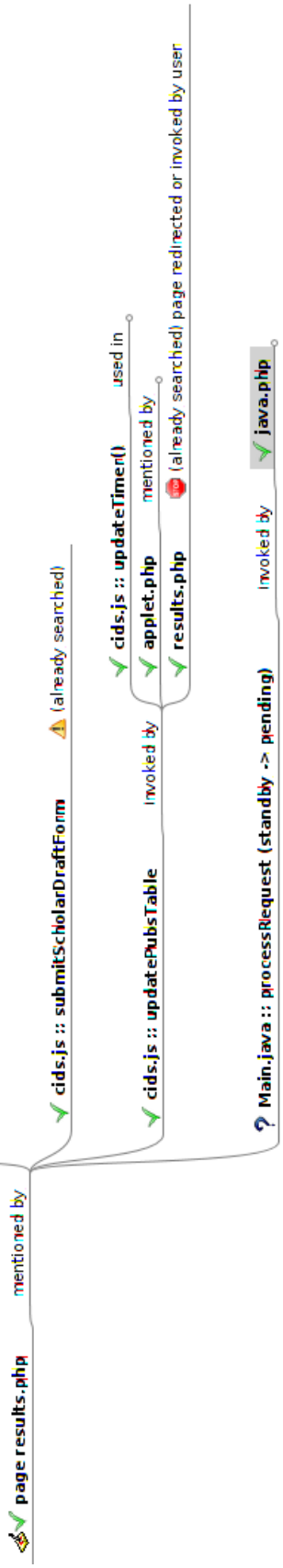


Figure B.2: Second image (of five) of the map of functions.

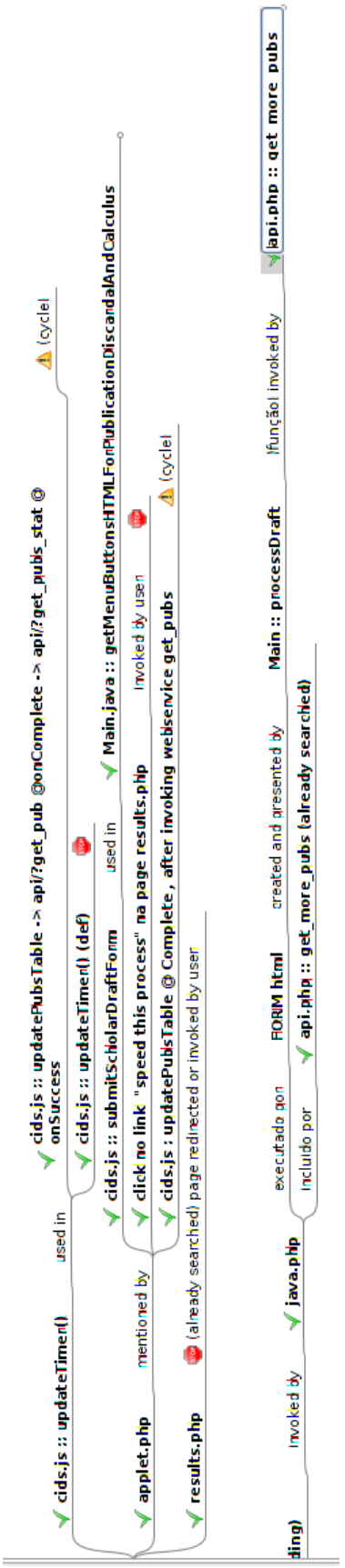


Figure B.3: Third image (of five) of the map of functions.

Figure B.4: Fourth image (of five) of the map of functions.

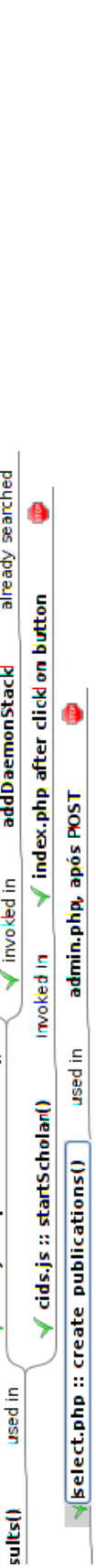


Figure B.5: Fifth image (of five) of the map of functions.

Appendix C

SQL Queries

This appendix shows the old and the new SQL queries responsible for calculating each bibliometric. These were developed during the development of the functionality "Data Verification".

Code C.1: Old SQL query for calculating the h-index of query of an author.

```
SELECT DISTINCT COUNT(p.p_id)
FROM query_pub AS q,publication AS p
WHERE q.a_id = $a_id  AND q.q_id = $_last_qid
AND p.p_id = q.p_id
AND q.discarded=
AND p.cits >= (SELECT DISTINCT COUNT(p2.p_id)
    FROM query_pub AS q2,publication AS p2
    WHERE q2.a_id = $a_id AND q2.q_id = $_last_qid
    AND p2.p_id = q2.p_id
    AND q2.discarded=
    AND (p2.cits>p.cits
    OR (p2.cits=p.cits AND p2.p_id>=p.p_id)));
```

Code C.2: New SQL query for calculating the h-index of query of an author.

```
SELECT DISTINCT COUNT(p.p_id)
    FROM query_pub AS q,publication AS p  JOIN
    (select p.p_id, p.cits as cites
    from publication p JOIN query_pub qp ON p.p_id=qp.p_id  AND qp.
        a_id=$a_id AND qp.q_id=$_last_qid
    AND NOT EXISTS(SELECT * FROM citation c JOIN citation_query cq
        ON cq.citation_id=c.id AND cq.query_id=$_last_qid WHERE c.
        pub_id=p.p_id)
union
SELECT p.p_id, IF(citations.pub_id IS NULL,0,COUNT(citations.id)) as
     cits_ns
    FROM publication p
    JOIN query_pub qp ON p.p_id = qp.p_id
    LEFT JOIN (SELECT c.id,c.pub_id,cq.*
                    FROM citation_query cq
                    JOIN  citation c ON c.id=cq.citation_id AND cq.
                        discarded=    ) citations ON citations.
                        pub_id =p.p_id AND citations.query_id=
                        $_last_qid
```

```
13
14      WHERE p.p_id IN (SELECT DISTINCT c.pub_id  FROM citation c JOIN
            citation_query cq ON c.id=cq.citation_id AND cq.query_id=
            $_last_qid)
15 AND qp.a_id = $a_id
16 AND qp.q_id = $_last_qid
17 AND qp.discarded =
18 GROUP BY p.p_id) as all_cites ON p.p_id=all_cites.p_id
19 WHERE q.a_id = $a_id  AND q.q_id = $_last_qid
20 AND p.p_id = q.p_id
21 AND q.discarded=
22 AND all_cites.cites >= (SELECT DISTINCT COUNT(p2.p_id)
23                  FROM query_pub AS q2,publication AS p2 JOIN
24                      (select p.p_id, p.cits as cites
25                          from publication p JOIN query_pub qp ON p.
                               p_id=qp.p_id  AND qp.a_id=$a_id AND qp.
                               q_id=$_last_qid
26                          AND NOT EXISTS(SELECT * FROM citation c JOIN
                                citation_query cq ON cq.citation_id=c.
                               id AND cq.query_id=$_last_qid WHERE c.
                               pub_id=p.p_id)
27                      union
28                      SELECT p.p_id, IF(citations.pub_id IS NULL,0,
                            COUNT(citations.id)) as cits
29                          FROM publication p
30                          JOIN query_pub qp ON p.p_id = qp.p_id
31                          LEFT JOIN (SELECT c.id,c.pub_id,cq.*
32                                          FROM citation_query cq
33                                          JOIN  citation c ON c.id=cq.
                                              citation_id AND cq.
                                              discarded=    )
                                              citations ON citations.
                                              pub_id =p.p_id AND
                                              citations.query_id=
                                              $_last_qid
34
35                          WHERE p.p_id IN (SELECT DISTINCT c.pub_id
                               FROM citation c JOIN citation_query cq
                               ON c.id=cq.citation_id AND cq.query_id=
                               $_last_qid)
36                          AND qp.a_id = $a_id
37                          AND qp.q_id = $_last_qid
38                          AND qp.discarded =
39                          GROUP BY p.p_id) as all_cites2
40                      ON p2.p_id=all_cites2.p_id
41                  WHERE q2.a_id = $a_id
42                  AND q2.q_id = $_last_qid
43                  AND p2.p_id = q2.p_id
44                  AND q2.discarded=
45                  AND (all_cites2.cites>all_cites.cites
46                  OR (all_cites2.cites=all_cites.cites AND p2.p_id>=p.
                        p_id)));
```

Code C.3: Old SQL query for calculating the h-index discerning self-citations of query of an author.

```
SELECT DISTINCT COUNT(p.p_id)
FROM query_pub AS q,publication AS p
WHERE q.a_id = $a_id  AND q.q_id = $_last_qid
AND p.p_id = q.p_id
AND q.discarded=
AND p.cits_ns >= (SELECT DISTINCT COUNT(p2.p_id)
    FROM query_pub AS q2,publication AS p2
    WHERE q2.a_id = $a_id AND q2.q_id = $_last_qid
    AND p2.p_id = q2.p_id
    AND q2.discarded=
    AND (p2.cits_ns>p.cits_ns
    OR (p2.cits_ns=p.cits_ns AND p2.p_id>=p.p_id)))
```

Code C.4: New SQL query for calculating the h-index discerning self-citations of a query of an author.

```
SELECT DISTINCT COUNT(p.p_id)
FROM query_pub AS q,publication AS p JOIN
        (select p.p_id, p.cits_ns as cites
            from publication p JOIN query_pub qp ON p.p_id=qp.p_id
               AND qp.a_id=$a_id AND qp.q_id=$_last_qid
            AND NOT EXISTS(SELECT * FROM citation c JOIN
               citation_query cq ON cq.citation_id=c.id AND cq.
               query_id=$_last_qid WHERE c.pub_id=p.p_id)
        union
        SELECT p.p_id, IF(citations.pub_id IS NULL,0,COUNT(citations
            .id)) as cites
            FROM publication p
            JOIN query_pub qp ON p.p_id = qp.p_id
            LEFT JOIN (SELECT c.id,c.pub_id,cq.*
                            FROM citation_query cq
                            JOIN  citation c ON c.id=cq.citation_id
                               AND cq.discarded=    AND cq.type=
                                        ) citations ON citations.
                               pub_id =p.p_id AND citations.
                               query_id=$_last_qid

            WHERE p.p_id IN (SELECT DISTINCT c.pub_id  FROM citation
                c JOIN citation_query cq ON c.id=cq.citation_id AND
                cq.query_id=$_last_qid)
            AND qp.a_id = $a_id
            AND qp.q_id = $_last_qid
            AND qp.discarded =
            GROUP BY p.p_id) as ns_cites
    ON p.p_id=ns_cites.p_id
WHERE q.a_id = $a_id  AND q.q_id = $_last_qid
AND p.p_id = q.p_id
AND q.discarded=
AND ns_cites.cites >= (SELECT DISTINCT COUNT(p2.p_id)
    FROM query_pub AS q2,publication AS p2 JOIN
                    (select p.p_id, p.cits_ns as cites
            from publication p JOIN query_pub qp ON p.p_id=qp.p_id
               AND qp.a_id=$a_id AND qp.q_id=$_last_qid
            AND NOT EXISTS(SELECT * FROM citation c JOIN
               citation_query cq ON cq.citation_id=c.id AND cq.
```

```
                query_id=$_last_qid WHERE c.pub_id=p.p_id)
28          union
29          SELECT p.p_id, IF(citations.pub_id IS NULL,0,COUNT(citations
                .id)) as cites
30              FROM publication p
31              JOIN query_pub qp ON p.p_id = qp.p_id
32
33              LEFT JOIN (SELECT c.id,c.pub_id,cq.*
34                              FROM citation_query cq
35                              JOIN  citation c ON c.id=cq.citation_id
                                    AND cq.discarded=    AND cq.type=
                                                ) citations ON citations.
                                    pub_id =p.p_id AND citations.
                                    query_id=$_last_qid
36              WHERE p.p_id IN (SELECT DISTINCT c.pub_id  FROM citation
                    c JOIN citation_query cq ON c.id=cq.citation_id AND
                    cq.query_id=$_last_qid)
37              AND qp.a_id = $a_id
38              AND qp.q_id = $_last_qid
39              AND qp.discarded =
40              GROUP BY p.p_id) as ns_cites2
41          ON p2.p_id=ns_cites2.p_id
42      WHERE q2.a_id = $a_id AND q2.q_id = $_last_qid
43      AND p2.p_id = q2.p_id
44      AND q2.discarded=
45      AND (ns_cites2.cites>ns_cites.cites
46      OR (ns_cites2.cites=ns_cites.cites AND p2.p_id>=p.p_id)))
```

Code C.5: Old SQL query for calculating the g-index of a query of an author.

```
1  SELECT DISTINCT COUNT(p.p_id)
2  FROM query_pub AS q,publication AS p
3  WHERE q.a_id = $a_id  AND q.q_id = $_last_qid
4  AND p.p_id = q.p_id
5  AND q.discarded=
6  AND (SELECT DISTINCT SUM(p2.cits)-POW(COUNT(p2.p_id),2)
7      FROM query_pub AS q2,publication AS p2
8      WHERE q2.a_id = $a_id AND q2.q_id = $_last_qid
9      AND p2.p_id = q2.p_id
10     AND q2.discarded=
11     AND (p2.cits>p.cits
12     OR (p2.cits=p.cits AND p2.p_id>=p.p_id))) >=0
```

Code C.6: New SQL query for calculating the g-index of a query of an author.

```
1  SELECT DISTINCT COUNT(p.p_id)
2  FROM query_pub AS q,publication AS p JOIN
3      (select p.p_id, p.cits as cites
4          from publication p JOIN query_pub qp ON p.p_id=qp.p_id  AND
               qp.a_id=$a_id AND qp.q_id=$_last_qid
5          AND NOT EXISTS(SELECT * FROM citation c JOIN citation_query
               cq ON cq.citation_id=c.id AND cq.query_id=$_last_qid
               WHERE c.pub_id=p.p_id)
6      union
7      SELECT p.p_id, IF(citations.pub_id IS NULL,0,COUNT(citations.id)
           ) as cits_ns
```

```
8           FROM publication p
9           JOIN query_pub qp ON p.p_id = qp.p_id
10          LEFT JOIN (SELECT c.id,c.pub_id,cq.*
11                          FROM citation_query cq
12                          JOIN  citation c ON c.id=cq.citation_id AND
                                cq.discarded=    ) citations ON
                                citations.pub_id =p.p_id AND citations.
                                query_id=$_last_qid
13          WHERE p.p_id IN (SELECT DISTINCT c.pub_id  FROM citation c
                JOIN citation_query cq ON c.id=cq.citation_id AND cq.
                query_id=$_last_qid)
14      AND qp.a_id = $a_id
15      AND qp.q_id = $_last_qid
16      AND qp.discarded =
17      GROUP BY p.p_id) as all_cites ON p.p_id=all_cites.p_id
18  WHERE q.a_id = $a_id  AND q.q_id = $_last_qid
19  AND p.p_id = q.p_id
20  AND q.discarded=
21  AND (SELECT DISTINCT SUM(all_cites2.cites)-POW(COUNT(p2.p_id),2)
22      FROM query_pub AS q2,publication AS p2 JOIN
23          (select p.p_id, p.cits as cites
24              from publication p JOIN query_pub qp ON p.p_id=qp.p_id
                    AND qp.a_id=$a_id AND qp.q_id=$_last_qid
25              AND NOT EXISTS(SELECT * FROM citation c JOIN
                    citation_query cq ON cq.citation_id=c.id AND cq.
                    query_id=$_last_qid WHERE c.pub_id=p.p_id)
26          union
27          SELECT p.p_id, IF(citations.pub_id IS NULL,0,COUNT(citations
                .id)) as cits_ns
28              FROM publication p
29              JOIN query_pub qp ON p.p_id = qp.p_id
30              LEFT JOIN (SELECT c.id,c.pub_id,cq.*
31                              FROM citation_query cq
32                              JOIN  citation c ON c.id=cq.citation_id
                                    AND cq.discarded=    ) citations ON
                                    citations.pub_id =p.p_id AND
                                    citations.query_id=$_last_qid
33              WHERE p.p_id IN (SELECT DISTINCT c.pub_id  FROM citation
                    c JOIN citation_query cq ON c.id=cq.citation_id AND
                    cq.query_id=$_last_qid)
34          AND qp.a_id = $a_id
35          AND qp.q_id = $_last_qid
36          AND qp.discarded =
37          GROUP BY p.p_id) as all_cites2 ON p2.p_id=all_cites2.p_id
38      WHERE q2.a_id = $a_id AND q2.q_id = $_last_qid
39      AND p2.p_id = q2.p_id
40      AND q2.discarded=
41      AND (all_cites2.cites>all_cites.cites
42      OR (all_cites2.cites=all_cites.cites AND p2.p_id>=p.p_id))) >=0
```

old g-index ns

Code C.7: Old SQL query for calculating the g-index discerning self-citations of a query of an author.

```
1 SELECT DISTINCT COUNT(p.p_id)
```

```
FROM query_pub AS q,publication AS p
WHERE q.a_id = $a_id  AND q.q_id = $_last_qid
AND p.p_id = q.p_id
AND q.discarded=
AND (SELECT DISTINCT SUM(p2.cits_ns)-POW(COUNT(p2.p_id),2)
    FROM query_pub AS q2,publication AS p2
    WHERE q2.a_id = $a_id AND q2.q_id = $_last_qid
    AND p2.p_id = q2.p_id
    AND q2.discarded=
    AND (p2.cits_ns>p.cits_ns
    OR (p2.cits_ns=p.cits_ns AND p2.p_id>=p.p_id))) >=0
```

Code C.8: New SQL query for calculating the g-index discerning self-citations of a query of an author.

```
SELECT DISTINCT COUNT(p.p_id)
FROM query_pub AS q,publication AS p JOIN
    (select p.p_id, p.cits_ns as cites
        from publication p JOIN query_pub qp ON p.p_id=qp.p_id  AND
            qp.a_id=$a_id AND qp.q_id=$_last_qid
        AND NOT EXISTS(SELECT * FROM citation c JOIN citation_query
            cq ON cq.citation_id=c.id AND cq.query_id=$_last_qid
            WHERE c.pub_id=p.p_id)
    union
    SELECT p.p_id, IF(citations.pub_id IS NULL,0,COUNT(citations.id)
        ) as cits_ns
        FROM publication p
        JOIN query_pub qp ON p.p_id = qp.p_id
        LEFT JOIN (SELECT c.id,c.pub_id,cq.*
                        FROM citation_query cq
                        JOIN  citation c ON c.id=cq.citation_id AND
                            cq.discarded=    AND cq.type=          )
                            citations ON citations.pub_id =p.p_id
                            AND citations.query_id=$_last_qid
        WHERE p.p_id IN (SELECT DISTINCT c.pub_id  FROM citation c
            JOIN citation_query cq ON c.id=cq.citation_id AND cq.
            query_id=$_last_qid)
    AND qp.a_id = $a_id
    AND qp.q_id = $_last_qid
    AND qp.discarded =
    GROUP BY p.p_id) as all_cites ON p.p_id=all_cites.p_id
WHERE q.a_id = $a_id  AND q.q_id = $_last_qid
AND p.p_id = q.p_id
AND q.discarded=
AND (SELECT DISTINCT SUM(all_cites2.cites)-POW(COUNT(p2.p_id),2)
    FROM query_pub AS q2,publication AS p2 JOIN
        (select p.p_id, p.cits_ns as cites
            from publication p JOIN query_pub qp ON p.p_id=qp.p_id
                AND qp.a_id=$a_id AND qp.q_id=$_last_qid
            AND NOT EXISTS(SELECT * FROM citation c JOIN
                citation_query cq ON cq.citation_id=c.id AND cq.
                query_id=$_last_qid WHERE c.pub_id=p.p_id)
        union
        SELECT p.p_id, IF(citations.pub_id IS NULL,0,COUNT(citations
            .id)) as cits_ns
```

```
28              FROM publication p
29              JOIN query_pub qp ON p.p_id = qp.p_id
30              LEFT JOIN (SELECT c.id,c.pub_id,cq.*
31                              FROM citation_query cq
32                              JOIN  citation c ON c.id=cq.citation_id
                                    AND cq.discarded=    AND cq.type=
                                                ) citations ON citations.
                                    pub_id =p.p_id AND citations.
                                    query_id=$_last_qid
33              WHERE p.p_id IN (SELECT DISTINCT c.pub_id  FROM citation
                     c JOIN citation_query cq ON c.id=cq.citation_id AND
                     cq.query_id=$_last_qid)
34          AND qp.a_id = $a_id
35          AND qp.q_id = $_last_qid
36          AND qp.discarded =
37          GROUP BY p.p_id) as all_cites2 ON p2.p_id=all_cites2.p_id
38      WHERE q2.a_id = $a_id AND q2.q_id = $_last_qid
39      AND p2.p_id = q2.p_id
40      AND q2.discarded=
41      AND (all_cites2.cites>all_cites.cites #nao mudei o nome para
           ns_cites, so alterei o WHERE acima (mas esta correcto)
42      OR (all_cites2.cites=all_cites.cites AND p2.p_id>=p.p_id))) >=0
```

Code C.9: Old SQL query for calculating the number of cited papers of a query of an author.

```
1 SELECT DISTINCT COUNT(p.p_id)
2 FROM query_pub AS q,publication AS p
3 WHERE q.a_id = $a_id  AND q.q_id = $_last_qid
4 AND p.p_id = q.p_id
5 AND q.discarded=
6 AND p.cits>0
```

Code C.10: New SQL query for calculating the number of cited papers of a query of an author.

```
1 SELECT DISTINCT COUNT(p.p_id)
2 FROM query_pub AS q,publication AS p JOIN
3     (select p.p_id, p.cits as cites
4             from publication p JOIN query_pub qp ON p.p_id=qp.p_id
                  AND qp.a_id=$a_id AND qp.q_id=$_last_qid
5             AND NOT EXISTS(SELECT * FROM citation c JOIN
                  citation_query cq ON cq.citation_id=c.id AND cq.
                  query_id=$_last_qid WHERE c.pub_id=p.p_id)
6         union
7         SELECT p.p_id, IF(citations.pub_id IS NULL,0,COUNT(citations
              .id)) as cits_ns
8             FROM publication p
9             JOIN query_pub qp ON p.p_id = qp.p_id
10            LEFT JOIN (SELECT c.id,c.pub_id,cq.*
11                              FROM citation_query cq
12                              JOIN  citation c ON c.id=cq.citation_id
                                    AND cq.discarded=    ) citations ON
                                    citations.pub_id =p.p_id AND
                                    citations.query_id=$_last_qid
```

```
13              WHERE p.p_id IN (SELECT DISTINCT c.pub_id  FROM citation
                    c JOIN citation_query cq ON c.id=cq.citation_id AND
                    cq.query_id=$_last_qid)
14          AND qp.a_id = $a_id
15          AND qp.q_id = $_last_qid
16          AND qp.discarded =
17          GROUP BY p.p_id) as all_cites ON p.p_id=all_cites.p_id
18 WHERE q.a_id = $a_id  AND q.q_id = $_last_qid
19 AND p.p_id = q.p_id
20 AND q.discarded=
21 AND all_cites.cites>0
```

Code C.11: Old SQL query for calculating the number of citations of a query of an author.

```
1 SELECT DISTINCT SUM(p.cits)
2 FROM query_pub AS q,publication AS p
3 WHERE q.a_id = $a_id  AND q.q_id = $_last_qid
4 AND p.p_id = q.p_id
5 AND q.discarded=
```

Code C.12: New SQL query for calculating the number of citations of a query of a query of an author.

```
1 SELECT DISTINCT SUM(all_cites.cites)
2 FROM query_pub AS q,publication AS p JOIN
3 (select p.p_id, p.cits as cites
4              from publication p JOIN query_pub qp ON p.p_id=qp.p_id
                   AND qp.a_id=$a_id AND qp.q_id=$_last_qid
5              AND NOT EXISTS(SELECT * FROM citation c JOIN
                   citation_query cq ON cq.citation_id=c.id AND cq.
                   query_id=$_last_qid WHERE c.pub_id=p.p_id)
6          union
7          SELECT p.p_id, IF(citations.pub_id IS NULL,0,COUNT(citations
               .id)) as cits_ns
8              FROM publication p
9              JOIN query_pub qp ON p.p_id = qp.p_id
10             LEFT JOIN (SELECT c.id,c.pub_id,cq.*
11                             FROM citation_query cq
12                             JOIN  citation c ON c.id=cq.citation_id
                                   AND cq.discarded=    ) citations ON
                                   citations.pub_id =p.p_id AND
                                   citations.query_id=$_last_qid
13             WHERE p.p_id IN (SELECT DISTINCT c.pub_id  FROM citation
                   c JOIN citation_query cq ON c.id=cq.citation_id AND
                   cq.query_id=$_last_qid)
14         AND qp.a_id = $a_id
15         AND qp.q_id = $_last_qid
16         AND qp.discarded =
17         GROUP BY p.p_id) as all_cites ON p.p_id=all_cites.p_id
18 WHERE q.a_id = $a_id  AND q.q_id = $_last_qid
19 AND p.p_id = q.p_id
20 AND q.discarded=
```

Code C.13: Old SQL query for calculating the number of citations discerning self-citations of a query of an author.

```
SELECT DISTINCT SUM(p.cits_ns)
FROM query_pub AS q,publication AS p
WHERE q.a_id = $a_id  AND q.q_id = $_last_qid
AND p.p_id = q.p_id
AND q.discarded=
```

Code C.14: New SQL query for calculating the number of citations discerning self-citations of a query of an author.

```
SELECT DISTINCT SUM(all_cites.cites)
FROM query_pub AS q,publication AS p JOIN
        (select p.p_id, p.cits_ns as cites
            from publication p JOIN query_pub qp ON p.p_id=qp.p_id
                AND qp.a_id=$a_id AND qp.q_id=$_last_qid
            AND NOT EXISTS(SELECT * FROM citation c JOIN
                citation_query cq ON cq.citation_id=c.id AND cq.
                query_id=$_last_qid WHERE c.pub_id=p.p_id)
        union
        SELECT p.p_id, IF(citations.pub_id IS NULL,0,COUNT(citations
            .id)) as cits_ns
            FROM publication p
            JOIN query_pub qp ON p.p_id = qp.p_id
            LEFT JOIN (SELECT c.id,c.pub_id,cq.*
                            FROM citation_query cq
                            JOIN  citation c ON c.id=cq.citation_id
                                AND cq.discarded=    AND cq.type=
                                        ) citations ON citations.
                                pub_id =p.p_id AND citations.
                                query_id=$_last_qid
            WHERE p.p_id IN (SELECT DISTINCT c.pub_id  FROM citation
                 c JOIN citation_query cq ON c.id=cq.citation_id AND
                 cq.query_id=$_last_qid)
        AND qp.a_id = $a_id
        AND qp.q_id = $_last_qid
        AND qp.discarded =
        GROUP BY p.p_id) as all_cites ON p.p_id=all_cites.p_id
WHERE q.a_id = $a_id  AND q.q_id = $_last_qid
AND p.p_id = q.p_id
AND q.discarded=
```

Code C.15: Old SQL query for calculating the ratio citations per paper of a query of an author.

```
SELECT DISTINCT AVG(p.cits)
FROM query_pub AS q,publication AS p
WHERE q.a_id = $a_id  AND q.q_id = $_last_qid
AND p.p_id = q.p_id
AND q.discarded=
AND p.cits>0
```

Code C.16: New SQL query for calculating the ratio citations per paper of a query of an author.

```
SELECT DISTINCT AVG(all_cites.cites)
FROM query_pub AS q,publication AS p JOIN
        (select p.p_id, p.cits as cites
            from publication p JOIN query_pub qp ON p.p_id=qp.p_id
                AND qp.a_id=$a_id AND qp.q_id=$_last_qid
            AND NOT EXISTS(SELECT * FROM citation c JOIN
                citation_query cq ON cq.citation_id=c.id AND cq.
                query_id=$_last_qid WHERE c.pub_id=p.p_id)
        union
        SELECT p.p_id, IF(citations.pub_id IS NULL,0,COUNT(citations
            .id)) as cits_ns
            FROM publication p
            JOIN query_pub qp ON p.p_id = qp.p_id
            LEFT JOIN (SELECT c.id,c.pub_id,cq.*
                            FROM citation_query cq
                            JOIN  citation c ON c.id=cq.citation_id
                                AND cq.discarded=    ) citations ON
                                citations.pub_id =p.p_id AND
                                citations.query_id=$_last_qid
            WHERE p.p_id IN (SELECT DISTINCT c.pub_id  FROM citation
                 c JOIN citation_query cq ON c.id=cq.citation_id AND
                 cq.query_id=$_last_qid)
        AND qp.a_id = $a_id
        AND qp.q_id = $_last_qid
        AND qp.discarded =
        GROUP BY p.p_id) as all_cites ON p.p_id=all_cites.p_id
WHERE q.a_id = $a_id  AND q.q_id = $_last_qid
AND p.p_id = q.p_id
AND q.discarded=
AND all_cites.cites>0
```

Code C.17: Old SQL query for calculating the ratio citations per paper discerning self-citations of a query of an author.

```
SELECT DISTINCT AVG(p.cits_ns)
FROM query_pub AS q,publication AS p
WHERE q.a_id = $a_id  AND q.q_id = $_last_qid
AND p.p_id = q.p_id
AND q.discarded=
AND p.cits_ns>0
```

Code C.18: New SQL query for calculating the ration citations per paper discerning self-citations of a query of an author.

```
SELECT DISTINCT AVG(all_cites.cites)
FROM query_pub AS q,publication AS p JOIN
        (select p.p_id, p.cits_ns as cites
            from publication p JOIN query_pub qp ON p.p_id=qp.p_id
                AND qp.a_id=$a_id AND qp.q_id=$_last_qid
            AND NOT EXISTS(SELECT * FROM citation c JOIN
                citation_query cq ON cq.citation_id=c.id AND cq.
                query_id=$_last_qid WHERE c.pub_id=p.p_id)
        union
        SELECT p.p_id, IF(citations.pub_id IS NULL,0,COUNT(citations
            .id)) as cites
```

```
8               FROM publication p
9               JOIN query_pub qp ON p.p_id = qp.p_id
10              LEFT JOIN (SELECT c.id,c.pub_id,cq.*
11                              FROM citation_query cq
12                              JOIN  citation c ON c.id=cq.citation_id
                                   AND cq.discarded=    AND cq.type=
                                           ) citations ON citations.
                                   pub_id =p.p_id AND citations.
                                   query_id=$_last_qid
13              WHERE p.p_id IN (SELECT DISTINCT c.pub_id  FROM citation
                    c JOIN citation_query cq ON c.id=cq.citation_id AND
                    cq.query_id=$_last_qid)
14          AND qp.a_id = $a_id
15          AND qp.q_id = $_last_qid
16          AND qp.discarded =
17          GROUP BY p.p_id) as all_cites ON p.p_id=all_cites.p_id
18 WHERE q.a_id = $a_id  AND q.q_id = $_last_qid
19 AND p.p_id = q.p_id
20 AND q.discarded=
21 AND p.cits>0
```

Appendix D

Installation and configuration guide

D.1 Introduction

This document is a guide to help install the application CIDS on a server machine.

D.2 Server requirements

Here is a list of requirements, that the remainder of this guide expects you to have installed:

Operating System CentOS (www.centos.org/);

Java Virtual Machine (JVM)) Java from *SUN* (http://www.oracle.com/technetwork/java/index.html);

Stack Apache/MySQL/PHP I recommend XAMPP (http://www.apachefriends.org/en/xampp.html) since it is what this guide uses, though you can adapt the configurations if you know how. Additionally, you must have PEAR::Mail(http://pear.php.net/Mail) installed and configured to be able to send emails;

Database Administration I recommend Phpmyadmin for its ease of use (http://www.phpmyadmin.net/);

Internet Access If you connect to the internet using a proxy, you should have it set (if it is not already);

Make sure that it is indeed Java from *Sun* that you are using and not some open-source JVM, because that will not work. If you need to have multiple JVM installation on your machine, you must activate Java from *SUN* to the super user (usually *root*) as well as to the user to running the web-server (usually *apache2* or *www*).

D.3 Installing and configuring CIDS 3.1

The first thing to do is fetch existing source code from http://code.google.com/p/cids/ and put it the web-server *www/htdocs* folder. You can either put directly at the root of the web-server or inside a folder. After this, you must:

- Create a database called "cids_3_1" (or whatever you want) and memorize it since this name is going to be used in the next step;
- Execute *create_config.sh*, located at the root of the "cids" folder. This will create several several configuration files where needed. These files should be then modified to reflect the environment in which they are. The files created are (relative to root folder):

  config.php Database configuration of CIDS;

  api/daemon.cfg Java program configuration;

  api/mail_cfg.php The field "cc" and "to" sent by Java program;

  codeigniter/application/config/database.php *CodeIgniter*'s database configuration;

  codeigniter/application/config/config.php *CodeIgniter*'s main configuration. Adjust only the variables located at the top of the file. The "base_url" should point to the codeigniter installation via URL, not CIDS's root. Change the string "index.php" to "ci_index.php";

  codeigniter/application/config/email.php *CodeIgniter*'s email configuration, so that it can send emails;

- Make sure that exists an ".htaccess" file inside *CodeIgniter*'s folder. Additionally, apply the following modifications to the configuration file of your web-server (/etc/httpd/conf/httpd.conf):
  - Locate the entry "AllowOverride" and set it to "All";
  - Add the following line:
    `RewriteRule ^/<cids_root_folder>/codeigniter/(.*)$ /<cids_root_folder>/codeigniter/ci_index.php/$1 [L]`
    (there is a white space between the first `$` and the following `/`)
- Locate the hard-coded reference to CIDS in the script "js/citation.js" and change it to the URL that is used to access CIDS;
- Create a crontab/cronjob for the super user that executes the script "vip-script.php" from thirty to thirty seconds. The crontab should be similar to the following one:

```
*/30 * * * * cd <path-to-cids-folder> && /usr/sbin/php
vip-script.php
```

You also need to insert a entry inside the table "settings" in the database. The column "isAccessToGSBlocked" must have an entry with value equal to "y". This is used to know if Google Scholar is blocking requests and previous crontab is responsible for updating this entry;

- Set the permissions of the folders "api/backup_logs/" and "admin/team_jobs/" to read, write, and execute for all. If the folders do not exist, create them.

- Although there is already a .JAR file ready for being executed (located at "api/Signed-CidsJava31.jar") if the Java program needs modifications, here is what you must do:

  – Use *Netbeans* (http://netbeans.org/ - I strongly recommend the version 6.8 because previous .JAR was developed with that version) to open the project "JAVA_project" located at the root of the application; and rebuild the project;

  – Navigate to the "/JAVA project/jar/" and execute the "sign_jar.sh" file, which signs the the .JAR file and copies it to "api/", overwriting what already exists there.

  If it complains that it can not find "jar_signer" executable, you must put it the "$PATH" in order to work.

  If it complains that the certificate has expired, you can renew the certificate if you wish, but the the program will still work;

  – If the Java program is running, navigate to "api" folder and execute "restart_daemon.sh" to kill and launch the Java program;

After all this, restart the web-server that all modifications take place. If you need to modify the "api/daemon.cfg" or the Java program itself, do not forget to re-launch the Java program afterwards. If you want to know for sure if the Java program is executing in background, run the following shell command:

```
ps aux | grep -v grep | grep SignedCidsJava31
```

This will return the current running processes of Java program. Make sure that only one is executing.

There is also an administration panel which can be access by appending "admin" at CIDS base. If you go there, you are presented with a login form. In order to access it, you should add a user to the "diary_user" table in the database. Notice that the "password" column should be a password hashed with MD5 function. I recommend adding a user using the following SQL query (adjust accordingly):

```
INSERT INTO user_diary VALUES
```

```
('<your-username>',MD5('<your-password>'))
```

After this, you should be able to access the administration interface.

Lastly, I am going to mentioned a common error that you might encounter when using CIDS. If, when using CIDS, appears a error related to the cache and cookies, you should:

1. Go to browser's preferences panel and also Java preferences panel and clear the cache/cookies;
2. Click "Ok" on the pop-up window and then press "Back" on the browser;
3. Insert a different query and try again;

Unfortunately, this not works deterministically, so you will have to keep trying until it works.

After all the above steps, you should have a working CIDS on your machine.

Glossary

API	Application Programming Interface. A set of web-services used to access and/or modify the state of a application.
bibtex	A style-independent text-based file format for lists of bibliography items, such as articles, books, and theses.
CIDS	Citation Impact Discerning Self-Citations.
Citation Non-Self	A citation whose authors are not present on the publication it cites.
HTML	HyperText Markup Language.

Bibliography

[1] D.W. Aksnes. A macro study of self-citation. Scientometrics, 56(2):235–246, 2003.

[2] F. Couto, T. Grego, C. Pesquita, and P. Veríssimo. Handling self-citations using google scholar. 13(2), 2009.

[3] B. Cronin and L. Meho. Using the h-index to rank influential information scientistss. Journal of the American Society for Information Science and Technology, 57(9):1275–1278, 2006.

[4] L. Egghe. Theory and practise of the g-index. Scientometrics, 69(1):131–152, 2006.

[5] A.S. Gami, V.M. Montori, N.L. Wilczynski, and R.B. Haynes. Author self-citation in the diabetes literature. Canadian Medical Association Journal, 170(13):1925, 2004.

[6] E. Garfield. The impact factor and using it correctly. Der Unfallchirurg, 48(2):413, 1998.

[7] A. W. Harzing. Publish or perish. `http://www.harzing.com/pop.htm#metrics`.

[8] J. E. Hirsch. An index to quantify an individual's scientific research output. Proceedings of the National Academy of Sciences of the United states of America, 102(46):16569, 2005.

[9] K. Hyland. Self-citation and self-reference: Credibility and promotion in academic publication. Journal of the American Society for Information Science and Technology, 54(3):251–259, 2003.

[10] P. Jacso. Google scholar's ghost authors, lost authors, and other problems. Library Journal, 2009.

[11] C.D. Kelly and M.D. Jennions. The h index and career assessment by numbers. Trends in Ecology & Evolution, 21(4):167–170, 2006.

[12] G. Nunberg. Google's book search: A disaster for scholars. `http://chronicle.com/article/Googles-Book-Search-A/48245/`.

[13] N. Oswald. Bitesizebio: Does your h-index measure up? http://bitesizebio.com/articles/does-your-h-index-measure-up/, 2009.

[14] M. Schreiber. Self-citation corrections for the hirsch index. EPL (Europhysics Letters), 78:30002, 2007.